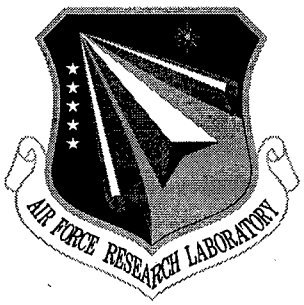
AFRL-IF-RS-TR-1999-62
Final Technical Report
April 1999



A COMPREHENSIVE APPROACH TO PLANNING AND SCHEDULING

University of Oregon

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. C641

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-1999-62 has been reviewed and is approved for publication.

APPROVED:

NORTHRUP FOWLER III
Project Engineer

FOR THE DIRECTOR:

EUGENE C. BLACKBURN, Chief
Information Technology Division
Information Directorate

If your address has changed or if you wish to be removed from the Air Force Research Laboratory Rome Research Site mailing list, or if the addressee is no longer employed by your organization, please notify AFRL/IFT, 25 Brooks Road, Rome, NY 13441-4505. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

A COMPREHENSIVE APPROACH TO PLANNING AND SCHEDULING

David W. Etherington
Matthew L. Ginsberg

Contractor: University of Oregon
Contract Number: F30602-95-1-0023
Effective Date of Contract: 27 June 1995
Contract Expiration Date: 27 June 1998
Short Title of Work: A Comprehensive Approach to Planning and Scheduling

Period of Work Covered: Jun 95 - Jun 98
Principal Investigator: David W. Etherington
Phone: (541) 346-0472

Matthew L. Ginsberg
(541) 346-0471

AFRL Project Engineer: Northrup Fowler III
Phone: (315) 330-3011

This research was supported by the Defense Advanced Research Projects Agency of the Department of Defense and was monitored by Northrup Fowler III, AFRL/IFT, 25 Brooks Road, Rome, NY 13441-4505.

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	April 1999	Final Jun 95 - Jun 98

4. TITLE AND SUBTITLE

A COMPREHENSIVE APPROACH TO PLANNING AND SCHEDULING

5. FUNDING NUMBERS

C - F30602-95-1-0023
PE - 62301E and 62702F
PR - C641
TA - 00
WU - 81

6. AUTHOR(S)

David W. Etherington and Matthew L. Ginsberg

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

University of Oregon
CIRL, 1269 University of Oregon
Eugene OR 97403-1269

8. PERFORMING ORGANIZATION REPORT NUMBER

N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Defense Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington VA 22203-1714

Air Force Research Laboratory/IFT
25 Brooks Road
Rome NY 13441-4505

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

AFRL-IF-RS-TR-1999-62

11. SUPPLEMENTARY NOTES

Air Force Research Laboratory Project Engineer: Northrup Fowler III/IFT/(315) 330-3011

12a. DISTRIBUTION AVAILABILITY STATEMENT

Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This project explored a new paradigm for planning based on, among other things, a clean separation of action selection and action sequencing (roughly, planning and scheduling). This approach provided several benefits, including more intuitive, powerful, and flexible planners and the ability to exploit the rapid advances in scheduling technology then occurring. At the same time, scheduling technology was advanced by enabling exploitation of structure inherent in the problems themselves.

The given separation, and generalizations thereof, were proven to be extremely powerful, and a number of planners now subscribe to the approach, including a mixed-initiative planner based on separating planning and scheduling that was developed and demonstrated through this effort. Moreover, the capabilities of scheduling systems were improved by several orders of magnitude through the research undertaken here.

The research effort uncovered two important principles. The first is the fact that compact, quantified representatives can be effectively exploited to dramatically increase the size of problems that can be tackled. The second is that solutions to realistic problems cluster in a variety of ways, and exploiting this clustering can provide significant computational as well as representational leverage.

14. SUBJECT TERMS

Planning, Scheduling, Search, Constraint Satisfaction, Lifted Representations, Solution Clusters, Approximate Planning, Symmetry

15. NUMBER OF PAGES

196

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Contents

1 Abstract

This project explored a new paradigm for planning based on, among other things, a clean separation of action selection and action sequencing (roughly, planning and scheduling). This approach provided several benefits, including more intuitive, powerful, and flexible planners and the ability to exploit the rapid advances in scheduling technology then occurring. At the same time, scheduling technology was advanced by enabling exploitation of structure inherent in the problems themselves.

The given separation, and generalizations thereof, were proven to be extremely powerful, and a number of planners now subscribe to the approach, including a mixed-initiative planner based on separating planning and scheduling that was developed and demonstrated through this effort. Moreover, the capabilities of scheduling systems were improved by several orders of magnitude through the research undertaken here.

The research effort uncovered two important principles. The first is the fact that compact, quantified representations can be effectively exploited to dramatically increase the size of problems that can be tackled. The second is that solutions to realistic problems cluster in a variety of ways, and exploiting this clustering can provide significant computational as well as representational leverage.

2 Executive Summary

At the time this project was proposed, existing generative planners such as SIPE or O-Plan bore more resemblance to programming languages or expert-system shells than to the search-based tools that are essential in real-time or mixed-initiative applications. This project constituted a multifaceted approach to addressing this difficulty.

Three main thrusts were envisioned: a substantial paradigm shift in AI's approach to planning and, through this paradigm shift, a major rethinking of the generative planning work throughout the ARPI; significant enhancements to scheduling (and constraint satisfaction) algorithms; and a general understanding of the relationship between planning and scheduling.

More technically, existing generative planning work had been hamstrung by a planning paradigm that lacked many intuitively essential features and was too inflexible for mixed-initiative planning. Work on scheduling had largely ignored the tremendous recent advances in constraint-satisfaction technology. Moreover, planning and scheduling research was too loosely connected, and the technical details of the interface between planning and scheduling had not been investigated. As a result integration had lagged, and planning research was not benefiting from improved solver capabilities.

This project aimed to address these difficulties by exploiting specific technical innovations that would allow significant advances in the state of the art in all three focus areas. In planning, the idea of approximate planning would be explored, providing a more natural paradigm, and allowing planning effort to be partitioned in ways that enhanced both computation and flexibility. In scheduling, existing constraint satisfaction (CSP) techniques would be enhanced to exploit the inherent problem structure that could allow CSP solvers to handle large realistic scheduling problems. Finally, developing a formal interface language between planning and scheduling would allow progress in scheduling to directly (and portably) impact planning efforts.

Set against these goals, the project has been very successful, although as the problems of planning and scheduling came to be better understood, the research evolved in ways that met the same goals in different ways.

It has become clear that planning *is* best treated as a decomposable process, and the decomposition of planning into action selection and action sequencing (and, more generally, constraint satisfaction) is becoming the approach of choice. A mixed-initiative planner based on this notion was developed and demonstrated. This planner captures many of the intuitions behind approximate planning, but in a more computationally attractive way.

A generalized framework for integrating planners and schedulers was developed based on a restricted variant of first-order logic, and planner-scheduler combinations based on this framework were demonstrated. The goal of general "plug-and-play" planners and schedulers has not been realized in the ARPI, mainly due to the fact that, in existing ARPI planners, planning and scheduling functions were so commingled as to make teasing them apart infeasible. Thus the general exploitation of the planner-scheduler separation waits for the next generation of planners to be built, but the technical advantages and disadvantages of the approach have been demonstrated.

Tremendous progress was also made in scheduling, resulting in several orders-of-magnitude

improvements in both feasible problem size and time required to solve. Somewhat paradoxically, some of these improvements came from representational insights gained in the process of developing the planner/scheduler interface language, and the realization that the efficiencies inherent in that representation could be exploited in the constraint solving process as well.

In the process of achieving the above, a number of key theoretical and practical insights were developed. Chief among these were the discovery of the existence and ubiquity of solution clusters and the realization that it is possible to use compact, "lifted", representations to dramatically improve the capabilities of problem solvers.

Lifted representations involve the use of quantified variables to express general facts (e.g., all F-16s have one engine) rather than explicitly listing all instances of those facts. Because lifted representations can be dramatically more compact, it is possible to address much larger problems without overflowing the capabilities of the machines involved. It was discovered that it is possible to use lifted representations without incurring the high computational overhead that previously led researchers to view these representations as impracticable. The computational savings from exploiting such representations have obvious applications throughout planning and scheduling.

It was also discovered that solutions to real problems tend to cluster. This "solution clustering" has far-reaching ramifications, from allowing the construction of more effective solvers, to techniques for identifying qualitatively different alternative plans and plan vulnerabilities, to the discovery of a new approach to finding robust plans.

The project also included a number of technology transfer initiatives, including ongoing efforts to transition constraint solving technology toward the USAF Air Mobility Command's aircraft flight planning system, and the spinoff of a startup company to exploit advances in scheduling.

3 Introduction

Two theoretical insights drove much of the progress on this project. The first is that complexity-class-based problem decomposition provides powerful technical leverage; the second is the discovery that solutions and partial solutions tend to cluster in subtle ways, and this clustering can be exploited to yield better solutions more quickly. Application of these two insights allowed advancement of the state of the art on a number of fronts.

3.1 Problem Decomposition

The effectiveness of problem decomposition was anticipated when this project was proposed, although it proved fruitful in a variety of unanticipated ways. In addition to exploiting the reduction of planning problems to scheduling problems, other reductions were found that aid in solving the resulting scheduling problems.

What is typically called "planning" actually consists of two distinct kinds of activity: *action selection*—deciding what to do—and *action sequencing*—deciding when to do it. The fundamental idea underpinning this project was that planning problems should be solved not by using a monolithic planner, but instead by first reducing them to scheduling

problems, and then by completing the solution using any of a variety of high-performance scheduling tools. This approach was proposed for three separate reasons:

1. A planning algorithm, *approximate planning* existed that worked by reducing a planning problem to an instance of NP.[1] This algorithm had a variety of attractive epistemological properties, such as the ability to naturally represent plans that were flexible in terms of admitting additional actions that were needed to achieve other goals.

2. It was natural to think of the "planner/scheduler interface" as the point at which the planning algorithm left off and the scheduling algorithm took over. This suggested that it would be possible to combine existing planning algorithms, such as O-Plan [Currie and Tate, 1991], and scheduling algorithms such as TACHYON [Stillman *et al.*, 1996].

3. As scheduling technology improved (and improving it was also part of this effort), planners could naturally be expected to exploit the improvements achieved.

Set against these three goals, the success of the project can be measured as follows:

1. The idea that planning problems should be solved by reducing them to instances of NP has been embraced by the planning community at large. In addition to approximate planning, Carnegie-Melon's GRAPHPLAN [Blum and Furst, 1995] planner works by first generating the search space associated with any particular problem and then searching using fairly standard techniques; AT&T's SATPLAN [Kautz and Selman, 1996] reduces a planning problem to one in propositional logic. In both cases, as in approximate planning, the reduction is not to a scheduling problem specifically, but to a class that is computationally equivalent to it. Neither this project nor the other authors found any effective way to achieve the reduction to scheduling directly, so that progress in scheduling (which is an NP problem) currently impacts planning technology only via the inclusion of that progress in NP tools generally.

2. A general, extensible, planner/scheduler interface framework was adopted. In the process, it was discovered that the representation language underlying the framework provides a useful compact representation, and algorithms that can exploit that compactness to improve their scalability and computational efficiency were developed.

   Demonstrations of combined planners and schedulers were developed in accord with the overall principles discussed above. Also as discussed above, the combinations work by reducing a planning problem to an instance of a problem in NP (using the interface framework developed under this award as an intermediate language); schedulers were then used to solve the associated search problem.

3. Much of the project's effort focused on the development of new scheduling technology, and the application of that technology to problems of practical interest. Specific new techniques developed include:

[1] A problem class is in *NP* if candidate solutions, once obtained, can be validated in time polynomial in the size of the problem description.

- Limited discrepancy search
- Schedule packing (discovered independently by McDonnell-Douglas)
- Squeaky wheel optimization

These techniques were applied to realistic problems provided by Boeing (aircraft assembly), Lucent (fiber-optic cable manufacture), and the USAF Air Mobility Command (flight routing), with excellent results.

3.2 Solution Clustering

Unlike complexity-based problem reduction, the idea of solution clustering was not anticipated at the start of the project, and the discovery that solutions and partial solutions tend to cluster, and that this clustering can be exploited, is another of the major achievements of this work.

Solution clusters are collections of (partial) solutions that are "near" each other according to some metric. For example, if a planned mission requires close air support, which can be provided in a number of ways (e.g., from a carrier in the Persian Gulf or an air base in Saudi Arabia), the basic plan together with all the different ways to provide the air support might constitute a cluster of solutions [Parkes, 1997].

The fact that there can be similar solutions to many problems should come as no surprise. What was surprising, however, was the discovery of the many different ways that solutions cluster, and how ubiquitous clustering behavior is. It was found that, even in random problems generated to explicitly avoid having the sort of structure one might expect to produce clusters of solutions, solutions do, in fact form large clusters.

A number of types of clustering were identified. In each case, members of a cluster share basic similarities, but these similarities may be with respect to:

solution space: members of a cluster share operational similarities (e.g., carrier- or land-based air support for the same mission)

abstract space: a cluster may be operationally diverse but related according to more abstract metrics (e.g., plans share the same target priority structure, but employ different mechanisms)

syntax: cluster elements are superficially distinct, but functionally indistinguishable (e.g., substituting different squadrons from the same base)

These notions of solution clustering contribute to much of the technical progress made over the life of the project, including squeaky wheel optimization, weighted limited discrepancy search, symmetry exploitation, supermodels, and model clustering, all of which are described later in this report.

A number of ways that clustering can be exploited were identified, including:

- reducing search-space size
- finding robust solutions

- identifying solution vulnerabilities
- effectively communicating solutions
- identifying distinct alternatives
- improving convergence toward optimality
- avoiding large, unproductive, search-space regions.

These are discussed in more detail in Section 6.2.

The remainder of this report describes each of the areas of progress, and focuses on the contributions of complexity-based decomposition and solution clustering. Section 7 describes applications of the research, and the final section briefly describes each of the major papers that were written as part of this project. Selected key papers are reproduced in the appendices.

4 Reduction of Planning to NP

It is appropriate to begin by discussing the difficulties and issues involved in translating planning problems into NP. In general, the approach taken by both this project and others has been to translate a particular planning problem into a constraint-satisfaction problem (CSP). This CSP can then be solved by any of a variety of specialized constraint satisfaction tools.

Unfortunately, blindly translating a planning problem into a constraint-satisfaction language typically fails to capture much of the structure of the original problem. As a result, the problems that are eventually presented to the CSP engine are frequently either intractably large or are solved inefficiently. Section 4.3 describes one possible solution to this difficulty, involving the use of CSP engines designed to work with non-ground (but still propositional) theories.

The difficulty caused by the large size of the CSPs produced by reducing planning problems to NP problems is also mitigated by the extensive progress that has been made in applying other complexity-based transformations to lower the cost of solving the resulting problem. Some of these techniques, including symmetry exploitation, are described in Section 6.1.

A final consequence of the reduction of planning to NP is that it may allow planning engines to capitalize on techniques that modify instances of NP so that solutions to the modified problem correspond to *robust* solutions to the original. By applying these techniques to the problems arising from planning problems, robust solutions to those problems can be developed. Further details on these techniques appear in Section 6.2.

4.1 Planning and constraint satisfaction

Since constraint-satisfaction problems are in NP, but planning problems are properly harder than NP, there can be no direct reduction of planning problems into NP; another way to see this is to realize that there are small planning problems (such as Towers of Hanoi) for which it is impossible to validate solutions in polynomial time because the solutions themselves are of length exponential in the size of the original problem.

At least four separate groups have attempted to reduce planning problems to CSPs, however, using four similar approaches. In chronological order, these approaches are the following:

1. Joslin's DESCARTES planner [Joslin and Pollack, 1995] turns a planning problem into not one CSP, but a sequence of CSPs. As it becomes necessary to add new actions to the plan, the size of the associated CSP grows. Development and refinement of DESCARTES continued under this project.
2. Ginsberg's approximate planning technique (developed under the scope of this project) [Ginsberg, 1995] introduces an auxiliary search space that is used to identify the actions that could contribute to the success or failure of the overall plan; a CSP is then constructed that analyzes interactions among these actions.
3. Blum and Furst's GRAPHPLAN [Blum and Furst, 1995] is similar; while approximate planning builds its search space from the goal backwards, GRAPHPLAN builds a search space by beginning with the given initial state and working forwards. After the modified search space is constructed, fairly conventional search techniques are used to find a solution in both cases.
4. Kautz, McAllester, Selman [Kautz *et al.*, 1996] and (independently) Bedrax [Bedrax-Weiss *et al.*, 1996] bound the length of the solution plan and then translate the associated instance of NP into a Boolean constraint-satisfaction problem. Kautz, McAllester and Selman produce the Boolean CSP by hand; Bedrax' approach (developed under this project) does so automatically but generally produces larger CSPs as a result.

4.2 Approximate Planning

Approximate planning deviates from traditional planning in its conception of what a "plan" is. Traditional planning considers a plan to be essentially a program—if all the plan steps are executed sequentially, the goal will be achieved. Conversely, approximate planning views a plan as a description of what must be done to achieve the goal—a description that could correspond to many execution sequences. Other things may be going on, but in most cases these will be irrelevant, and the plan will achieve the goal. Approximate planning is approximate precisely because *some* things that may happen along with the planned steps may break the the plan; if the set of such things is negligible with respect to the set of possible executions of the plan, the plan is approximately correct.

For example, when executing a plan to fly into enemy territory passing designated way points and strike a target, the pilot certainly does other things, such as check fuel consumption and watch for enemy aircraft, that are not specified in the plan. He may even do significant unplanned things, such as opportunistically strike a mobile missile launcher he flies over. Of course, there are things he could do that would jeopardize the mission (e.g., expend his armaments en route), but by and large the basic plan he was given suffices—it is approximately correct.

Because approximate plans do not completely specify everything that can happen, they allow important plan manipulations, such as planning for subgoals separately and accepting user specifications for (partial) subplans, to happen in natural ways. Approximate plans (either from separate subgoals or from mixed-initiative specifications) can be merged together

to produce combined plans. Since plans are not detailed programs, they can typically be interleaved or concatenated without difficulty—the actions from one fitting into the spaces that were implicitly there in the other. Where there *are* conflicts, the resulting merged plan is refined using the same planning mechanism, until the result is once again approximately correct vis à vis the combined set of constraints.

The formal language originally developed to describe approximate plans was found to present a variety of technical problems, including the possibility that the results of the planning process, in some situations, might not be suitable for future refinement by the approximate planner: plan descriptions could become exponentially long, and certain classes of plans could not be described at all. This would have made it difficult to apply the formalism to mixed-initiative planning. These difficulties led to a search for a specific language in which approximate plans could be described. The goal here was to develop a language with favorable complexity properties that would support the type of separation of different-complexity problem features that came to be seen as essential.

The particular desiderata were that the language have the following properties:

a. It should be possible to construct instances of the language in polynomial time from the plan-space search graph.
b. Questions involving sentences in the language (e.g., are two sentences approximately equal?) should have complexity properly lower than the complexity of the original planning problem from which they originate.

The first property is important because it makes it possible to overcome the difficulties that had been plaguing attempts to implement an approximate planner, specifically the fact that an NP-complete problem (subgraph isomorphism) needed to be solved at every node in the search space. The second is important because it provides an indication that the approximate planning paradigm is actually being used to separate the portion of the problem that is NP-hard (i.e., the scheduling part) from the portion that is more difficult (the action selection part).

This search was successful, and a language that represents potential solutions to action-selection problems in isolation was developed, with action-sequencing problems corresponding to the determination of whether a particular sentence in the language is equivalent to the empty plan. This determination is NP-complete and corresponds to solving the action-sequencing (i.e., scheduling) component of any particular planning problem. Planning problems themselves are in general EXP-space complete, which is indeed harder than NP-complete.

The modified language that was developed avoids the difficulties described above, and specific algorithms were developed that cleanly separate the two phases of planning. All of this work is summarized in the *KR'96* paper, "A new algorithm for generative planning" [Ginsberg, 1996b], which is included in the appendix.

The refinement process involved in developing an approximate plan suggested an application of the idea of "bounding" used in approximate knowledge compilation. In particular, when the planner was faced with an unexpanded subgoal that (for whatever reason) it was not prepared to expand, it could make a variety of assumptions regarding the eventual

expansion. If it made pessimistic assumptions, the resulting set of plans returned might be too restrictive; if the assumptions were optimistic, the returned set might be too liberal.

This idea led to a generalization of approximate planning—indeed of conventional planning, too. Approximate planning returns the plan set that holds whenever the optimistic and pessimistic assumptions produce approximately the same result. Conventional generative planners return the first plan that achieves the goal under maximally pessimistic assumptions. As the optimistic and conservative bounds converge together, they provide progressively more accurate characterizations of the actual set of plans. An anytime planning algorithm can use the plans in the optimistic set when it is forced to return a plan, but the distance between the optimistic and pessimistic sets might be usable to characterize the planner's "confidence" in the result. Moreover, whenever the planner notices that the optimistic expansion of a plan set has become empty, there is no need to expand further—there can be no expansions that will achieve the goal—and the search space can be pruned, presumably producing significant savings of effort.

An approximate planner was implemented, based on the new representation language and incorporating the notions of progressively-tightening bounds described above. The planner was interfaced to a CIRL scheduler that makes sequencing decisions (described below), and was demonstrated during a site visit and at the AIPS meeting in Edinburgh. The planner was built to support mixed-initiative protocols, with the user able to select various nodes for expansion, determine if nodes will eventually be solvable by a scheduler, and so on.

4.3 Lifted solvers

All of the reductions of planning to CSPs discussed in the previous section work by modifying the given planning problem so that it is in NP, and then translating the instance of NP into a language that is equivalent to propositional logic. This is because most existing efficient search engines, such as WSAT [Selman *et al.*, 1993] and TABLEAU [Crawford and Auton, 1996] require fully-grounded theories; for example, if the problem specification required an F-18 for some task, this would have to be replaced with the disjunction of all possible individual F-18s. A downside of this translation is that much of the structure of the original NP instance is lost; when a single domain axiom such as

$$\forall x.\ \text{F-117A}(x) \supset \mathit{stealthy}(x)$$

is replaced by all of its ground instances, the single principle underlying the axiom itself is no longer accessible to the solver.

There are two difficulties with using fully-grounded theories. The first is that the size of the resulting theory can be enormous, since each axiom is replaced by d^n ground instances, where d is the domain size and n is the number of variables in the non-ground (or "lifted") axiom. Secondly, the lifted axiom may reflect an important truth about the domain. As an example, suppose that a particular problem can be shown to require an aircraft that is either supersonic or stealthy; a particular attack helicopter H_1 can be eliminated because it is neither.

In the ground case, the reasoning leading to the elimination of the helicopter will refer to ground facts about the domain and about the helicopter in question; the crucial domain facts

$$\neg supersonic(H_1) \wedge \neg stealthy(H_1)$$

will have been replaced with uninformative ground tokens such as

$$\neg L3768 \wedge \neg L3221 \ .$$

Because of this, it will be impossible to generalize the same argument to conclude that attack helicopters in general are unsuitable; instead, the analysis will need to be repeated in its entirety for each helicopter. As a result, the reasoner will spend significant time effectively revisiting the same decision many times.

These difficulties provide a compelling argument for the use of lifted axiomatizations when the conversion to NP is performed. Prior to the work performed in this project, however, there had been two accepted reasons why this was impossible:

1. The introduction of variables changes the problem from propositional to first-order logic, and first-order logic is known to be undecidable.
2. There is a substantial overhead in dealing with lifted formulations, since variable bindings need to be retained and analyzed, resolution needs to include a unification step, and so on. The success of ground engines was due in no small part to extremely efficient implementations that were believed to be incapable of absorbing additional costs of this sort.

Ways around both of these difficulties were found.

The first solution is the simplest: instead of extending the language to full first-order logic, it was extended only to the case where all of the variables are required have values selected from predefined, finite, sets. The resulting language (QPROP, for Quantified Propositional Logic) is formally equivalent to propositional logic, but the allowance of quantification made it possible to capture the structure of a problem more naturally than ground axioms permitted, and this turned out to be profoundly important. While QPROP was not new, the realization that it can be used for efficient computation was.

The solution to the second problem was more subtle. Ground search engines spend the bulk of their time searching for database axioms satisfying particular properties. In a non-systematic engine such as WSAT [Selman *et al.*, 1993], for example, each primitive search operation requires the selection of a random unsatisfied axiom, and a list of all such axioms must be maintained as the algorithm proceeds. Systematic approaches such as RELSAT [Bayardo and Schrag, 1997] or TABLEAU [Crawford and Auton, 1996] spend the bulk of their time unit propagating, or computing the forced consequences of variable assignments. (This is often called *forward checking* by the non-Boolean CSP community.) Unit propagation involves finding all database axioms containing at most one satisfiable literal, once again involving a search through the database.

These database searches typically proceeded by enumerating and then examining every axiom impacted by the most recent variable assignment; in some sense, these searches themselves proceeded using generate and test. Since the databases in question can be quite

large, it is important that the search for axioms satisfying specific properties be conducted using more efficient methods, and these methods require the exploitation of the database structure that is reflected in the QPROP axiomatization—which is lost in the translation to a ground representation. The search savings associated with using the non-ground language can more than offset the constant-factor costs of dealing with the variables. This was verified experimentally with WSAT, and a prototype implementation was developed for systematic methods as well.

Two other advantages of working within the QPROP representation were also identified, both a direct result of the fact that a QPROP axiomatization is typically orders of magnitude smaller than its ground counterpart. First, for many realistic problems, the ground description is intractable by virtue of its size alone; a problem that is computationally easy in theory but involves 10^9 axioms is likely to be anything but easy in practice. Second, the reduced QPROP axiomatization is likely to fit into memory cache in situations where the ground axiomatization is forced to reside in main memory. Since access times are so much shorter for cache than for main memory, substantial performance improvements result here as well. Both of these effects were verified experimentally using the lifting of WSAT to QPROP developed under this award.

The work with QPROP formulations exploited the structure present in realistic problems. One motive for this was that planning problems can be naturally described in this way, and the success of the QPROP work can therefore be expected to have a significant impact on planning generally.

With ground solvers, earlier work showed that some encodings of planning problems are infeasible simply because the size of the fully-ground theory grows much too rapidly with the size of the planning problem for anything but toy problems to ever be manageable. Lifted solvers can manipulate many more clauses than would be feasible in a ground solver, since a lifted solver makes use of a far more compact representation. Working with lifted constraint formulae can, in effect, handle large numbers of propositional constraints en masse. The compactness of the lifted representation also allows the avoidance of much of the swapping that would be incurred were one working with the much larger propositional representation (which rapidly becomes too large to fit in memory), thus helping with scaling.

A preliminary implementation of a nonsystematic, lifted solver solved "pigeonhole" problems for which the size of the propositional theory was on the order of 100 megabytes, but for which the lifted representation is at least two orders of magnitude smaller. This prototype lifted solver allowed some of the causal encodings of planning problems that were infeasible with ground solvers to be reconsidered.

It was also found that there are classes of natural, but redundant, lifted clauses that can help focus the search engine. The size of the ground representations of these clauses has made them impractical for ground solvers, and hence not useful, in the past, but their lifted representations remain compact. An advantage of this approach is that such constraints can correspond to decisions that might be hard-coded into a special-purpose solver, but in this case they are still represented declaratively and so are solver-independent [Jónsson and Ginsberg, 1996].

Symmetries in lifted theories

Symmetries in problems arise from, among other things, the existence of interchangeable resources. From a problem-solving point of view, symmetries can be extremely difficult to detect, but failure to detect them can result in exponential increases in the size of the space that must be searched in order to find a solution. New techniques for detecting and breaking symmetries to reduce the complexity of reasoning problems were developed. Most of this work is described in Section 6.1.1. Only the fact that excellent progress has also been made in generalizing these techniques to lifted problem solvers is discussed here.

An algorithm for finding symmetries in the descriptions of planning problems was developed, and a theorem was proved presenting conditions guaranteeing the preservation of those symmetries when a constraint satisfaction problem (CSP) is generated from the problem description. This algorithm was implemented, and it was shown experimentally, using a CSP encoding of a logistics planning domain, that finding symmetries in the lifted representation yields a significant improvement in overall performance when compared to finding symmetries in the ground theory. This is in turn is significantly better than current solution techniques that do not exploit symmetry (see Figure 1).[2] A paper describing this work, from which Figure 1 was taken, was published at AAAI-97 [Joslin and Roy, 1997], and is included in the appendix.

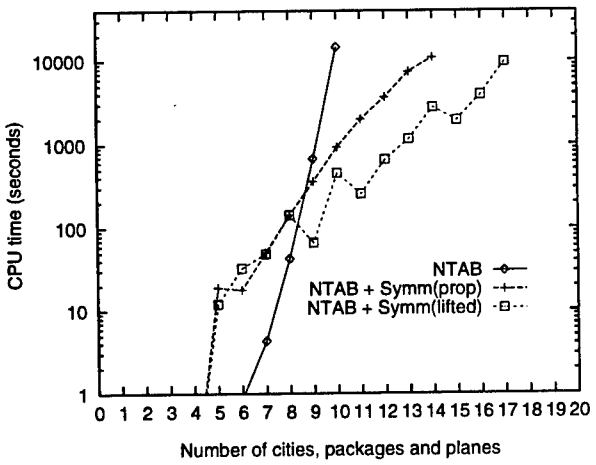


Figure 1: Effectiveness of lifted symmetry breaking for a logistics problem. (Note that the y-axis is log scaled.)

[2]NTAB is an implementation of TABLEAU [Crawford and Auton, 1996], a state-of-the-art propositional theorem-prover.

5 Planner/Scheduler Interface

As was mentioned earlier, one of the original motivations for this project was to exploit the inherent differences between planning and scheduling, letting planners and schedulers do what they each do best. In order to maximally exploit progress in each area, it was proposed that a general-purpose planner-scheduler interface framework would be developed, which would allow arbitrary planners to communicate with arbitrary schedulers.

It turns out that, for historical reasons, most existing planners are not set up to take advantage of such a framework. Since the distinction between planning and scheduling, and the importance of separating them, was not widely recognized when they were designed, the scheduler-like functionality is not cleanly separated in a way that would allow it to be replaced by calls to other schedulers. That being said, substantial progress was made on planner-scheduler interfaces which can be used in designing new planners or retrofitted into existing planners. Both types of applications were demonstrated.

The interface framework developed allows for the kind of clean separation between planners and schedulers that has been argued to be necessary. It also provides the flexibility necessary to let planners utilize schedulers in whatever way they are best able. The overall approach taken here of solving planning problems by reducing them to scheduling, to the extent that it can be used to understand the actions taken by existing planning engines, suggested a natural way in which such planners could be linked to scheduling tools: first, find the point at which the "planning" (i.e., action selection) stops and "scheduling" (i.e., action sequencing) begins, and then replace a particular tool's action-sequencing methods with those of a dedicated scheduler.

This was accomplished for two separate planner/scheduler combinations. The first involved CIRL technology on both sides, reducing the output from an approximate planner to a problem that could be solved using either a Boolean satisfiability tool or a dedicated CIRL scheduler. The second effort involved a similar interface between the University of Edinburgh's O-Plan planner [Currie and Tate, 1991] and TACHYON [Stillman *et al.*, 1996], a temporal-constraint management tool developed at GE. In both cases, QPROP was used as the mediating language between the two processes.

The flexibility provided by QPROP here was important, since it was often useful for the planner and scheduler to interact before the problem reduction to NP was complete. The mechanism supports passing a fully-fleshed-out plan to a scheduler which would sequence the plan's actions, or querying a scheduler about the feasibility of satisfying some set of constraints particular to a single node in the planner's search. As an example, a planner might use a scheduling engine to check a subplan for consistency even before the overall plan was complete. If the subplan failed, QPROP could be used to pass a nogood back from the scheduler to the planner, indicating the nature of the difficulty.

Future planners will likely want to take full advantage of the dichotomy between planning and scheduling by cleanly separating the tasks and using a language such as QPROP to pass information between them. However, since it isn't convenient for most existing planners to use a scheduling engine to search for a complete solution, but consistency checking can still be useful, this architecture provides an opportunity to reap some of the benefits in the interim. The use of this sort of logical language appears to be within the reach of most

ARPI planners, including SIPE (SRI) [Wilkins, 1988], O-Plan (Edinburgh), DESCARTES (U Pitt/CIRL), and approximate planning (CIRL).

Having translated the problem into QPROP, it would ideally be solved using either a dedicated scheduling tool or a general-purpose tool capable of working with the lifted axioms directly. No such general-purpose tool existed when the demonstration systems were built, and instead a compiler (CSPC) [Jónsson, 1996] was used that grounded the QPROP axiomatization while identifying specific structures (such as temporal orders) that could be passed to specialized modules (such as TACHYON) for analysis.

QPROP's flexibility in this area suggests that it may be a viable alternative to SPAR [SPAR Working Group, 1996] as a language for describing planning and scheduling problems. While both languages provide a degree of representational flexibility without committing to domain specific primitives, only QPROP does so within the widely accepted framework of traditional logic, in a form that is compositional, and with a clearly defined semantics and known computational complexity.

6 Scheduling Technology

Much of the technical progress made during this project consisted of advances in scheduling (CSP and satisfiability) technology. These advances fall into two broad classes: heuristic improvements and epistemological discoveries.

Section 6.1 discusses heuristic improvements, leading to an enhanced ability to solve scheduling problems. These techniques can themselves be split into systematic methods (Section 6.1.1) and nonsystematic methods (Section 6.1.2). Section 6.2 discusses epistemological discoveries, where a better understanding of the nature of the problem being solved was achieved. The contributions here include a completely novel definition of robustness that has attractive complexity-theoretic properties and also appears to be a close match to operational needs, as well as a new approach to the identification of operationally distinct solutions and their vulnerabilities, and insights into potential ways to improve convergence of optimization algorithms on good solutions.

6.1 Heuristic improvements

6.1.1 Systematic methods

The systematic methods that were investigated during this project fall into two groups: symmetry exploitation and extensions to limited discrepancy search (LDS).

Symmetry exploitation: A symmetry of a problem consists of a set of choices that yield isomorphic solutions. For example, choices among identically equipped squadrons stationed at the same airfield might constitute a symmetry of an air campaign planning problem. If the target can't be hit by the 34th fighter squadron, it can't be hit by the 4th FS, either. Moreover, if the 34th can do the job, there is no virtue in expending effort trying to see if the 4th can do it better. Since there may be many sets of symmetric choices in a problem,

a scheduler can be forced to explore a combinatorial number of options that effectively correspond to the same solution (or nonsolution).

A group-theoretic characterization was developed of a broad class of symmetries that can appear in problems. Ways to use symmetries, once they are recognized, to generate additional axioms that rule out all but one of the isomorphic solutions were demonstrated. These "symmetry breaking" axioms force the search engine to consider only one of the ways the symmetric choices could be made, thus significantly pruning the search space. Intuitively, whole classes of isomorphic solutions are collapsed to a single representative.

While the general problem of detecting and breaking all symmetries is intractable, partial methods—ways to break enough symmetries to significantly simplify the problem—in polynomial time were studied. It was shown that, in several cases, a polynomial effort can break the symmetries fully or partially. Once again, the goal was to find ways to use polynomial effort to reduce the size of an NP-complete problem.

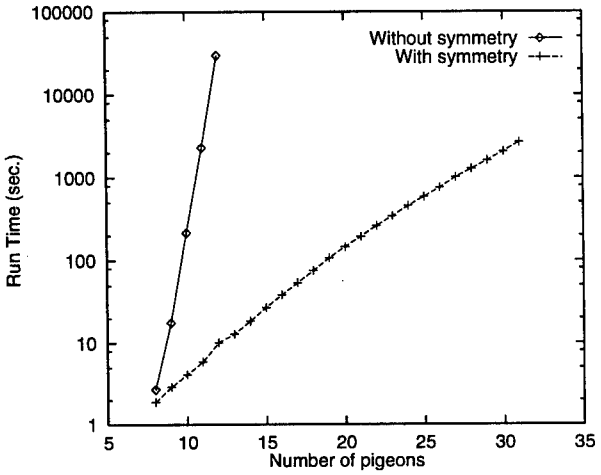


Figure 2: Run times for the pigeonhole problem with and without symmetry. Note that the y axis is log scaled.

These techniques were implemented in a system that takes a propositional satisfiability problem, detects symmetries in the theory, and "compiles" out a set of axioms that break the symmetries. These axioms do in fact help to constrain the search space. The approach was tested on toy problems, including the "pigeonhole" problem, where it reduces the time needed to prove that $n+1$ pigeons cannot fit in n holes from exponential (in n) to linear (see Figure 2), and the n-queens problem, where it leads to substantial computational speedups over existing search-based methods (see Figure 3). These symmetry algorithms were also applied to the NP problems arising from planning, with similar results. This work was described in detail in a paper published at *KR'96* [Crawford *et al.*, 1996], which is included in the appendix, and the figures are taken from there.

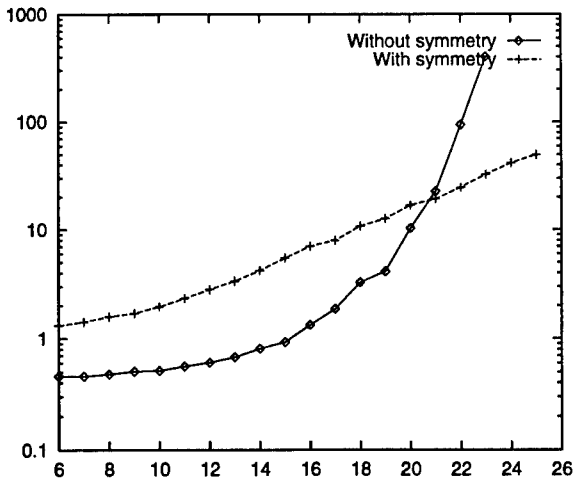


Figure 3: Average run times over 50 random permutations of n-queens with and without symmetry. Note that the y-axis is log scaled.

Weighted Discrepancy Search: As part of an earlier ARPI award, limited discrepancy search (LDS) [Harvey and Ginsberg, 1995] was developed and evaluated. That work showed that LDS is an effective technique for avoiding "early mistakes" that otherwise may prevent a depth-first search engine from solving problems effectively. However, LDS itself is limited in its inability to exploit fine-grained heuristic information. To deal with this shortcoming, LDS was extended to "weighted discrepancy search", or WDS.

In LDS, a heuristic identifies the "best" task to move next as a schedule is constructed. LDS traverses the search space in a way that minimizes the number of violations of the heuristic, but for choice points where there is more than one way to violate the heuristic, LDS does not try to pick the best one. In WDS, nodes in a search space are explored not in order of the increasing *number* of deviations from the search heuristic, but in order of a weighted sum of such deviations. This allows the heuristic to be followed more closely when the heuristic preference is stronger, when previous experience has shown the heuristic to be more accurate, or other indicators suggest the heuristic estimate to be more reliable.

Formulae were derived giving the expected cost incurred by WDS in searching for a solution given a heuristic probability of success at each node. These formulae depend on certain simplifying assumptions made about the probabilistic interdependencies among the search nodes; both these assumptions and the formulae themselves were tested experimentally, and found to be valid for trees of substantial depth, such as those arising in realistic search problems. The cost formula can be used to obtain an optimal set of weights for WDS.

It was shown that WDS extends both LDS and depth-first search, as well as iterative sampling (a nonsystematic search procedure that performs well on some scheduling in-

stances), from which it follows that a search protocol based on these optimal weights will outperform the previous methods. Preliminary validation of this result was obtained by showing that WDS outperforms LDS on the McDonnell-Douglas benchmarks (`http://www.NeoSoft.com/~benchmrx` which are related to scheduling aircraft manufacturing. The performance of WDS on benchmark problems from the OR library (`http://mscmga.ms.ic.ac.uk/info.html`) also showed that this general-purpose approach is competitive with specialized algorithms developed elsewhere [Bedrax-Weiss, 1999].

6.1.2 Nonsystematic methods

Schedule packing (SP) (elsewhere referred to as "Doubleback Optimization") [Crawford, 1996] is an optimization technique that was developed as part of an earlier ARPI award. This project improved SP in a variety of ways. An understanding of the fundamental mechanism that makes SP work was also developed, leading to the development of a broad new class of nonsystematic search methods applicable to scheduling problems based on generalizations of that mechanism. This class of techniques is referred to as "squeaky wheel optimization" (SWO) [Joslin and Clements, 1998].

Improvements to schedule packing: The basic idea in schedule packing is that a preliminary "seed" schedule can be used to help identify those task chains that are sources of difficulty for the scheduler. That information is then used by the scheduler to determine the priority with which tasks should be addressed in the next attempt. While this technique has been shown to be a powerful tool in constructing near-optimal schedules for large industrial problems, it has limitations. More specifically, SP can only be applied to scheduling problems whose sole objective is to minimize the time required to perform the specified activities. Real problems' objective functions are typically far more complex.

There is also very limited feedback between successive SP passes. Experimental results show that SP frequently gets one part or another of a problem "right", but these partial solutions do not inform each other, making it hard for the system to get the whole problem right. Exploiting this understanding SP's strengths and weaknesses, the algorithm was improved and generalized.

It was also shown that introducing a small amount of randomness to SP (i.e., violating the suggested priority ordering occasionally) can yield significant improvements, although these improvements are highly sensitive to the degree of randomness. These findings are consistent with those for other types of local search, where controlled randomness is an important factor in finding good solutions. This discovery helped to explain a variety of observed effects in terms of small changes in the algorithm leading to significant performance changes. Somewhat surprisingly, a "simulated annealing" approach, in which the degree of randomness is gradually decreased, was found not to be effective.

In examining the schedules generated by SP, it was observed that problems often tend to split into fairly natural subproblems. Some of these subproblems can be hard, and stochastic solvers will only rarely find good subschedules. In attempting to schedule the entire problem, it is therefore unlikely that all of the subproblems will simultaneously be scheduled well. This suggests taking a good schedule, splitting it up, and separately attempting

to improve each portion, using existing algorithms might provide leverage on particularly difficult problems. This is different from standard divide-and-conquer in that a good solution is itself needed in order to define the division into subproblems. This technique was applied with striking effectiveness to difficult manufacturing scheduling problems, but no automatic methods for choosing good partitionings were developed.

All of these techniques were also applied to a set of fiber-optic cable production line scheduling problems, derived from realistic data provided by the manufacturer. SP in isolation is not an appropriate tool for solving this problem because the objective function is a weighted sum of several factors. The problems also use constraints that are more complex than could be handled by the current SP implementation.

Squeaky Wheel Optimization: In an effort to understand how to generalize SP to apply to such problems, an analysis of what SP does was performed. This analysis led to an understanding that SP consists of the following phases:

1. First generate a sequence of jobs, with higher "priority" jobs being earlier in the sequence. The priorities for subsequent passes are constructed using feedback from the analysis phase of the algorithm.
2. Given a prioritization, SP constructs a schedule greedily, first scheduling the highest priority jobs.
3. Given a schedule, an analysis phase suggests which jobs are "trouble spots" on the assumption that they should have been prioritized more highly when the initial sequence was constructed.[3]

This process is then repeated to produce a new candidate schedule; the overall process is described as "squeaky wheel" optimization (SWO) because the squeaky wheel (in step 3) gets the grease (in step 1 of the next iteration). The basic ideas underlying the approach are as follows:

1. Good solutions can reveal problem structure. Analysis of a good solution often identifies elements of that solution that work well, and elements that work poorly. A resource that is used at full capacity, for example, may represent a bottleneck. This information about problem structure is local, in the sense that it may only apply to some part of the search space currently under examination, but may be extremely useful in helping figure out in what direction the search should go next.

2. Local search can benefit from the ability to make large, coherent moves. It is well known that local search techniques tend to become trapped in local optima, from which it may take a large number of moves to escape. Random moves are a partial remedy, and in addition, most local search algorithms periodically just start over with a new random assignment. While random moves, small or large, are helpful, was hypothesized that the SWO architecture works, in part, because of its ability to also make large *coherent* moves. A small change in the priorities (step 1) may correspond

[3]In SP, this analysis is trivial, based only on jobs' relative position in the schedule.

to a large change in the schedule generated in step 2. Swapping the priorities of two tasks may change the positions of those two tasks in the schedule and, in addition, allow some of the lower priority tasks, later in the sequence, to be shuffled around to accommodate those changes. This is a large move that is "coherent" in the sense that it is similar to what might be expected from moving the higher priority task, then propagating the effects of that change by moving lower priority tasks as well. This single move may correspond to a large number of moves in a search algorithm that only looks at local changes to the schedule, and may thus be difficult for such an algorithm to find.

Note that this is a general architecture, and not itself a specific algorithm. SP can be viewed as an instance of this architecture, as can the OPTIFLEX scheduler [Syswerda, 1994], a highly successful commercial product. These schedulers may appear to have little in common, but it appears that principles that underlie both approaches have been uncovered. In the case of the OPTIFLEX scheduler, for example, this led to the hypothesis that it is not genetic algorithms *per se* that made the scheduler so effective, but rather the manner in which prioritization and greedy construction were combined.

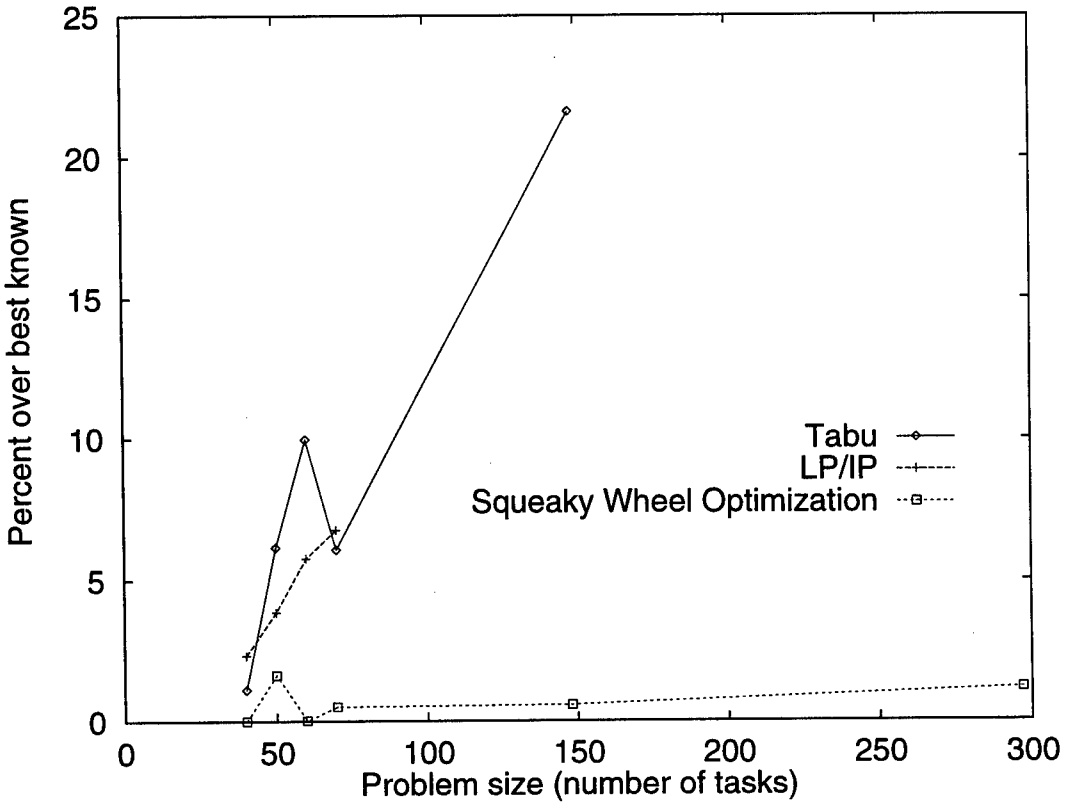


Figure 4: Performance of squeaky wheel optimization

SWO was applied to two problem domains. The first is the fiber-optic cable manufacturing problem that led to the development of the method; here, SWO is the only known

method capable of producing viable solutions at all on the largest problem instances provided by the manufacturer. On these problems, its performance is far superior to that of algorithms developed by the operations research community to solve identical problems, as shown in Figure 4,[4] which compares the performance of SWO and two well-known techniques from the operations research (OR) community, Tabu search [Glover and Laguna, 1997], and a linear/integer programming approach combining MINTO [Nemhauser *et al.*, 1994] and CPLEX [CPLEX Optimization, Inc., 1996]. The x-axis gives the number of cables being produced, and the y-axis compares the solution found using the given method to the best solution known.

In an attempt to understand the breadth of the applicability of the technique, it was applied to graph coloring as well. Graph coloring was chosen because it is clearly distinct from scheduling, and because implementing SWO in that domain was fairly straightforward. SWO is competitive with many graph-coloring-specific algorithms (it is better than the specialized Iterated Greedy algorithm, while worse than the specialized IMPASSE algorithms). It is substantially better than the other general purpose techniques against which it was tested (including TABLEAU, WSAT and GSAT).

A portable, Java-based, visualization tool that demonstrates the application of SWO to manufacturing problems was developed, and the technique was demonstrated on cable production problems in the system demonstration track at *AAAI-98.*

6.2 Epistemological improvements: Solution Clustering

Most work on CSPs focuses on the search for solutions to those CSPs, which are often called "models." By extending this idea, it is possible to construct solutions that have specific desirable properties. A key development arising from this project was the notion of "solution clusters." A solution cluster is a set of solutions that are near each other, in the sense that it is computationally easy to move among them. By identifying clusters of solutions, planners can gain robustness, isolate distinct alternatives, and identify critical solution features. Two distinct but related notions of clustering, supermodels and model clusters, were identified.

A *supermodel* is a model with the property that small changes can be accommodated by other small changes. If a supermodel is "broken," it can be fixed without much effort or disruption. Supermodels exist for many problems, and the complexity of repair of small disruptions was shown to be polynomial. Moreover, the complexity of finding a supermodel was shown to be unchanged from that of simply finding a model. This suggests that finding a supermodel when the original problem is solved will be worthwhile, since it ensures a tractable repair problem.

A *model cluster* is a set of related models that share a common structure. It is defined by a set of "control" parameters whose values are fixed in such a way that the remaining parameters only loosely constrain the solution. Thus, if the control parameters remain stable, situational change will be easy to accommodate. The settings for the control variables determine the critical elements of a solution: solutions with different settings are qualita-

[4]Taken from [Joslin and Clements, 1998].

tively different. All the models in a cluster represent the same basic solution, although they may differ in their peripheral details.

Four main applications for model clusters in planning and scheduling systems were identified. First, clusters represent qualitatively different solutions: different clusters represent real, operationally distinct, solutions. Secondly, the control parameters of a cluster represent limiting factors of the solution; if these change, confidence in the solution should drop dramatically. By identifying such limiting factors in advance, planners can assess and communicate vulnerabilities in solutions. These can then be protected or circumvented. Thirdly, clusters are robust solutions: changes to the noncontrol parameters are easily dealt with by other models in the cluster. Finally, clustering behavior amongst solutions (and nonsolutions) can often be exploited to guide search algorithms to better solutions more quickly.

Model clusters were shown to frequently exist and algorithms were developed to identify some control variables. The following subsections describe in more detail the ramifications of the discoveries about clustering made during this project.

6.2.1 Robustness

Supermodels and model clusters give complementary notions of robustness: finding a supermodel ensures that the chosen solution can be adapted to deal with change in polynomial time, while a model cluster automatically accommodates variability in the noncontrol parameters. In both cases, solving a somewhat more difficult problem ensures that the schedule produced can be modified in polynomial time to respond to situational changes at execution time. The rationale for this is that there is likely to be more time available to develop a course of action than there will be to adjust it when the unpredictable happens.

It was shown that supermodels and model clusters exist for some theories, and that supermodels can be adapted to small changes in polynomial time. It was also shown that it is possible to augment problem descriptions with constraints that require solutions found to be supermodels, so that finding a supermodel is NP-complete, and thus in the same complexity class as finding any model at all. Experimental results indicated that finding supermodels exhibits the same sort of "phase transition" behavior that occurs in the search for models. This is important because it suggests that it will be possible to search for supermodels using the "branch and bound" techniques that are currently used to apply satisfiability techniques to optimization problems.

Preliminary algorithms were developed that identify the control variables for model clusters, which appear to be related to short prime implicants (minimal entailed clauses) of the problem. Experiments were performed indicating that a combination of the two techniques may be profitable, with model clustering used to identify control variables, after which the supermodeling algorithms can be applied to the remaining variables to produce more general and flexible results.

One fundamental upshot of these observations is the conclusion that robustness is a property of plans, not of algorithms: a plan is not robust because a smart planner will be able to fix it in time to respond to changes, but because the plan itself anticipates those changes. This fact appears to have been overlooked in prior work. A paper on this work

appeared at *AAAI-98* [Ginsberg *et al.*, 1998b], and is included in the appendix.

6.2.2 Improving convergence

Deeper insights into the structure of problems, and the implications of this structure for the design of effective solvers were gained throughout the project. One type of structure is the "phase transition" that occurs in many problems as parameter variation moves problem instances from the "mostly satisfiable" region to the "mostly unsatisfiable" region. The boundary between these two regions is often extremely narrow, and although the phase transition at this boundary is widely recognized as an important feature of optimization search spaces, little is known about the deeper structure of the boundary region. It was shown experimentally that within the boundary region, exponentially large clusters of solutions develop, and that these clusters have compact representations. Similarly, exponentially large regions exist corresponding to "failed clusters": for each failed cluster, there is a single clause in the theory that, if eliminated, would turn the failed cluster into a cluster of models. The exponentially large regions that arise from failed clusters contain no models, but have many assignments with just one violated constraint. Hence, these regions are very likely to trap local search procedures, and their exponential size is likely to render standard tabu search methods useless.

This insight explains one way local search procedures get trapped, but it also may help to identify techniques for helping local search escape from such traps. The fact that the failed clusters have simple representations means that if these representations can be found, it may be feasible to exclude the clusters. A paper on this work appeared at *AAAI-97* [Parkes, 1997], and is included in an appendix.

Failed local search attempts were studied, and preliminary evidence found that they contain information that could be used in preprocessing to direct search into areas that are more likely to succeed and/or be useful. It was found that preprocessing to reduce a problem to its "hard kernel" can be highly effective. The idea is to do just a subset of all of the possible preprocessing (which, for large problems, can itself be very time consuming), but to ensure that the work done will be helpful to the subsequent search with a systematic or nonsystematic solver.

7 Technology Transfer

This section describes specific technology transfer efforts that demonstrated the applicability of the ideas developed under this award to realistic problems.

7.1 Boeing

Limited discrepancy search, WDS, and schedule packing were all applied to a problem related to aircraft manufacture provided by McDonnell-Douglas (now Boeing). This problem involves nearly 600 tasks using 17 resources and contains thousands of constraints. CIRL technology produces construction schedules some 5–10% shorter than alternative methods.

In addition, CIRL's technology can produce solutions as good as the best alternative methods in seconds, and solutions within 5% if the best schedules known in minutes.

7.2 Lucent

Squeaky wheel optimization is the clear method of choice on Lucent's problems in fiber-optic cable manufacture, outperforming both tabu search [Glover and Laguna, 1997], and a linear/integer programming approach combining MINTO [Nemhauser *et al.*, 1994] and CPLEX [CPLEX Optimization, Inc., 1996]. The other methods are incapable of finding legal schedules on problems involving more than 150 cables; even on smaller problems, SWO generally finds schedules of substantially higher quality. The best solutions known on the larger problems have been found by using SWO to generate candidate solutions and then refining these solutions using OR techniques.

7.3 AMC

Extensive interactions were conducted with John Lemmer (Rome Labs), Capt. Ray Burke and Maj. Mark Weiser (then USAF Air Mobility Command), and various individuals at AMC's contractor, NCI, Inc., to investigate the applicability of CIRL scheduling technology to AMC's Advanced Computer Flight Planning problem. A prototype implementation was developed that includes:

- Integration of the existing C5 performance polynomials.
- Full knowledge of the weather. All climbs and cruises are done with interpolated weather data. The prototype probably models the effects of weather more accurately than the production system in use at AMC.
- A great circle route calculator. This produces a great circle route between any two points that uses the weather data to calculate the route's cost and time. The great circle route initially climbs as high as possible, and then cruises until enough fuel has been burned to climb higher.
- Bounded Dijkstra and A* least cost algorithms to search for optimal routes. Both algorithms search in 3-dimensional space.

The prototype system was used to evaluate the improvements to route construction time and solution quality that can be achieved by applying more sophisticated search technology to these routing problems. This system was used to evaluate a number of alternative search strategies for the problem, and several that outperform the existing production system were identified.

7.4 Microsoft Project

An interface was built between the CIRL scheduler and Microsoft PROJECT, a tool widely used in the industrial community to represent scheduling problems (although not to solve those problems optimally). The challenge here was to build a robust translator between the CIRL's description of resource-constrained scheduling problems and that used by Microsoft

(MPX format). Most of these problems were solved successfully, and a large scheduling benchmark problem was encoded in PROJECT. The PROJECT encoding was translated automatically to the CIRL scheduling language, passed to the CIRL scheduler, optimized, and then returned to PROJECT for further display or analysis.

This interface makes CIRL's ARPI-developed scheduling technology accessible to a wide range of military and industrial users.

8 Publications

This section lists the main publications arising from this project, with a brief description of each. The most relevant papers among these are reproduced in the appendix, and are flagged with a †.

1. "A new algorithm for generative planning" [Ginsberg, 1996b]†
   Existing generative planners have two properties that one would like to avoid if possible. First, they use a single mechanism to solve problems both of action selection and of action sequencing, thereby failing to exploit recent progress on scheduling and satisfiability algorithms. Second, the context in which a subgoal is solved is governed in part by the solutions to other subgoals, as opposed to plans for the subgoals being developed in isolation and then merged to yield a plan for the conjunction. This paper presented a reformulation of the planning problem that appears to avoid these difficulties, describing an algorithm that solves subgoals in isolation and then appeals to a separate NP-complete scheduling test to determine whether the actions that have been selected can be combined in a useful way. Appeared at *KR'96.*

2. "Symmetry breaking predicates for search problems" [Crawford *et al.*, 1996]†
   Many reasoning and optimization problems exhibit symmetries. Previous work has shown how special purpose algorithms can make use of these symmetries to simplify reasoning. Here, a general scheme was presented for exploiting symmetries by adding "symmetry-breaking" predicates to the theory. This approach can be used on any propositional satisfiability problem, and can be used as a pre-processor to any (systematic or nonsystematic) reasoning method. The paper discussed methods for generating partial symmetry-breaking predicates, and shows that, while the general case of adding symmetry-breaking axioms appears to be intractable, in several specific cases symmetries can be broken either fully or partially using a polynomial number of predicates. These ideas have been implemented and experimental results are included on two classes of constraint-satisfaction problems. Appeared at *KR'96.*

3. "An approach to resource constrained project scheduling" [Crawford, 1996]†
   This paper surveyed the algorithms used in the CIRL scheduler, including double-back optimization and the application of LDS to scheduling. The results are quite good: CIRL has achieved the best known results on scheduling problems of realistic size and character. Appeared in the DARPA-supported *Artificial Intelligence and Manufacturing Research Planning Workshop.*

4. "Partition search" [Ginsberg, 1996c]
   While dependency-directed backtracking is a key concept in solving constraint-satisfaction problems, it has found little application in adversarial contexts, such as game-tree search. In essence, this paper shows how dependency-directed backtracking can be used in game-tree search. Experimental results show that partition search gives significant computational leverage over traditional alpha-beta pruning. Appeared at *AAAI-96.*

5. "Is "early commitment" in plan generation ever a good idea?" [Joslin and Pollack, 1995]
   Action selection is a critical step in planning. In particular, it is the step that makes planning harder than scheduling. When a planner determines that more actions are needed to achieve its goals, it can add the constraint that one of a set of candidate actions must occur, but postpone deciding which until the decision is forced (least commitment), or it can explore one candidate and only consider the other choices if necessary (early commitment). This paper discussed the tradeoff between least commitment and early commitment in action selection, and proposed a balanced approach that was shown to be superior to past methods. Appeared at *AAAI-96.*

6. "Tuning local search for satisfiability testing" [Parkes and Walser, 1996]†
   Local search is currently the method of choice for many academic, military, and industrial search problems. Unfortunately, local search algorithms tend to have many parameters that must be tuned for each problem class. This paper presented a probabilistic method for automatically optimizing one of the most important of these parameters in a way that makes efficient use of the work invested. The method was used to study the performance of WSAT, one of the best local search algorithms, on a hard class of satisfiability problems. Extensive experimental results were obtained over a much wider range of problem sizes than previously feasible. Appeared at *AAAI-96.*

7. "Toward efficient default reasoning" [Etherington and Crawford, 1996]
   This paper proposed a computationally viable approach to approximate consistency checking in default reasoning, a critical step in implementing formally grounded default reasoning systems. While cast in the context of default reasoning, the ideas are generally applicable in situations where a quick but approximate answer to an intractable question is required. Appeared at *AAAI-96.*

8. "Do computers need common sense?" [Ginsberg, 1996a]
   This paper made and defended three claims: (1) It is incumbent on the knowledge representation and nonmonotonic reasoning communities to demonstrate that their ideas will eventually lead to improvements in the performance of implemented systems. (2) A reasonable working definition of "commonsense" reasoning is that it is the process of using polynomial techniques to convert a large instance of an NP-hard problem to a smaller instance on which search techniques can be applied effectively. (3) It is a consequence of these first two claims that the most pressing problem facing the commonsense community is the identification of realistic problems and problem struc-

tures for which commonsense reductions are both necessary and effective. Appeared at *KR'96.*

9. "Efficient Reasoning using Procedures" [Jónsson and Ginsberg, 1996]
   For many constraint satisfaction problems, there are well known, fast algorithms and functions that solve parts of the problem. Using these methods directly to solve the subproblems significantly speeds up the solving process. Unfortunately, doing this has usually required the solver to be changed, or the correctness criteria to be re-examined. A solid formal model was developed of the use of procedural reasoning in solving constraint satisfaction problems, the common technique whereby a declarative inference engine can allow certain results to be computed procedurally. This turns out to be quite a subtle issue, requiring both formalization of procedural attachment in terms of an ability to extend partial solutions to a CSP and formalization of search in terms of a function on a set of possible successor nodes. The model described here provides a general mechanism for using procedures with almost any search engine, in such a way that it is easy to add any procedures without changing the engine. Furthermore, the framework allows proving conditions that are sufficient to guarantee systematicity and completeness for the search engines using those procedures. These ideas are being incorporated to extend the functionality of CIRL's declarative reasoning tools. This paper appeared in *KR'96.*

10. *The CSPC compiler: Users' manual* [Jónsson, 1996]†
    The CSPC compiler takes constraint satisfaction problems and transforms them into propositional satisfiability problems suitable for input into a variety of solvers. CSPC is based on the QPROP representation language.

11. "Exploiting symmetry in lifted constraint satisfaction problems" [Joslin and Roy, 1997]†
    When search problems have large-scale symmetric structure, detecting and exploiting that structure can greatly reduce the size of the search space. Previous work has shown how to find and exploit symmetries in propositional encodings of constraint satisfaction problems (CSPs). This paper considered problems that have more compact "lifted" (quantified) descriptions from which propositional encodings can be generated. An algorithm for finding symmetries in lifted representations of CSPs was described, and sufficient conditions were shown under which these symmetries can be mapped to symmetries in the propositional encoding. Using two domains (pigeonhole problems, and a CSP encoding of planning problems), it was experimentally demonstrated that the approach of finding symmetries in lifted problem representations provides a significant improvement over previous approaches that find symmetries in propositional encodings. Appeared at *AAAI-97.*

12. "Clustering at the phase transition" [Parkes, 1997]†
    The region in which theories are on the verge of being unsatisfiable is of recurrent interest because of its natural connection with optimization problems. Even in this region, however, it is quite common that if a problem has any solutions, then it

has many solutions. Real constraint problems have structure which can be exploited (and removed) to simplify finding solutions. The natural question is whether, when all available structure in a problem's constraints has been exploited, there is any structure left. This paper showed that even a problem that apparently has no structure at all in its constraints can still have a great deal of structure in its solution space. This structure takes the form of a clustering of solutions, and has repercussions on the performance of solvers. In particular, this clustering gives some insight into why the phase transition region is so hard even for local search, and may suggest ways to improve local search performance. Appeared at *AAAI-97*.

13. "What does knowledge representation have to say to AI?" [Etherington, 1997]
In recent years, the subarea of Knowledge Representation and Reasoning (KR) has become more and more of a discipline unto itself, focusing on artificial problems while other areas of AI have tended to develop their own representations and algorithms. There are signs that this is changing, however. This paper abstracts a talk that explored what the current state of KR has to offer to AI, and ways to increase KR's relevance. Abstract of invited talk that appeared at *AAAI-97*.

14. "Modern Planning and Scheduling Technologies" [Drabble, 1998]
A number of successful applications which have been developed using modern planning and scheduling techniques are described. The applications cover a number of areas of current business interest and include manufacturing, logistics, and crisis planning. In addition to the techniques and benefits described through the applications, several other planning and scheduling techniques are introduced and described.

15. "Repairing Plans on the Fly" [Drabble *et al.*, 1997]
Even with the most careful advance preparation, and even with in-built allowance for some degree of contingency, plans must be altered to account for execution circumstances and changes of requirements. Methods for repairing plans to account for execution failures and changes in the execution situation have been developed. This paper described the algorithms used for plan repair in O-Plan and gave an example of their use. Appeared at the *JPL Workshop on Planning and Scheduling Systems for Space*, October 1997.

16. "Propositional STRIPS Planning Problem Reversal" [Massey, 1998]
This paper argued that a long-held assumption about Propositional STRIPS-Style Planning (PSSP), namely that there is an inherent directional asymmetry in PSSP independent of particular problems or planning algorithms, is illusory. To show this, a simple, tractable construction was given that for any PSSP problem P produces a "reverse problem" P^R by reversing each action in the domain of P. Plans in P were shown to be precisely isomorphic to reversals of plans for P^R. Since backward planning in P^R corresponds to forward planning in P and vice-versa, this technique also provided a simple way to reverse the directionality of traditional planning algorithms.

17. "Workflow Support in the the Air Campaign Planning Process" [Drabble *et al.*, 1998b]
The aim of this paper was to describe a model developed to describe the Air Cam-

paign Planning (ACP) process and to characterize the capabilities of systems and technologies being developed to support this process within the DARPA/Rome Laboratory Planning Initiative (ARPI). Verb/noun(s)/qualifier(s) statements were used to define a process ontology that details the activities that take place in generating ACP plans (e.g. refine, issue, analyze) a set of process products flowing through the process, and the capabilities of agents and systems. The paper described the motivation behind developing the verb/noun(s)/qualifier(s) model, defined the different entities in the model, and showed how the model can be used to define the capabilities and process plans steps of the ACP process and in particular the USAF's Air Campaign Planning Tool (ACPT). The model has been mapped to the Task Formalism domain description language of O-Plan and used to validate an approach of using O-Plan as "planning to plan" workflow aid for Acpt. Details of the experiments used to validate the approach were provided. The paper concluded with details of further work which aimed to improve the reactive and scheduling capabilities of the system. Presented at the Fourth International Conference on Artificial Intelligence Planning Systems 1998 (AIPS '98) workshop on Collaborative Planning, Pittsburgh, June 8th, 1998.

18. "Modeling and Supporting the Air Campaign Planning Process" [Drabble *et al.*, 1998a]
    This paper outlined and motivated a model developed to describe the Air Campaign Planning (ACP) process and to characterize the capabilities of systems and technologies being developed to support this process within the DARPA/Rome Laboratory Planning Initiative (ARPI). Verb/noun(s)/qualifier(s) statements were used to define a process ontology which details the activities which take place in generating ACP plans, a set of process products flowing through the process, and the capabilities of agents and systems. The paper showed how the model can be used to define the capabilities and process plans steps of the ACP process, as well as in systems integration architectures for intelligent workflow generation, replanning and repair. Invited, and under review, for *Knowledge Engineering Review.*

19. "Makespan Scheduling for Assembly Tasks: Extended Abstract" [Drabble and Clements, 1997]
    A number of applications in the space domain require complex assembly planning and scheduling. For example, assembly, integration and verification, satellite command sequencing and satellite assembly. One of the main objectives in each of these applications is to identify a minimum length schedule or makespan which accomplishes all the activities in the task. The aim of this article was to provide a description of some of the techniques being developed at CIRL to produce schedules with minimum makespan. The article described the ways in which these techniques could be used for applications such as assembly, integration and verification. Appeared at *The JPL Workshop on Planning and Scheduling Systems for Space*, October 1997.

20. "Can search play a role in practical applications?" [Ginsberg *et al.*, 1998a]†
    This paper argued that received wisdom about the ineffectiveness of search in fielded AI applications is due not to the overwhelming size of the search spaces involved,

but to pathologies of certain search algorithms. A number of general techniques for overcoming these pathologies were described, and shown to make search feasible for real problems of practical interest. Appeared at *AI Meets the Real World '98,* University of Connecticut, Stamford, CT, September 1998.

21. "A non-deterministic semantics for tractable inference" [Crawford and Etherington, 1998]
    Even with the extensive improvements in search technology seen in recent years, it will always be necessary to reason with problems that are too large for complete solution methods. This article described a reasoning mechanism that is guaranteed to be tractable, if incomplete, but has a well-defined semantics delimiting the types of inference which it sanctions. This semantics was generalized to a family of semantics of increasing deductive power and computational complexity. Appeared at the *AAAI-98,* July 1998.

22. "Supermodels and Robustness" [Ginsberg *et al.*, 1998b]
    When search techniques are used to solve a practical problem, the solution produced is often brittle in the sense that small execution difficulties can have an arbitrarily large effect on the viability of the solution. The AI community has responded to this difficulty by investigating the development of "robust problem solvers" that are intended to be proof against this difficulty. This paper argued that robustness is best cast not as a property of the problem solver, but as a property of the solution. A new class of models for a logical theory, called supermodels, was introduced that captures this idea. Supermodels guarantee that the model in question is robust, and allow quantification of the degree to which it is so. The theoretical properties of supermodels were investigated, showing that finding supermodels is typically of the same theoretical complexity as finding models. A general way to modify a logical theory so that a model of the modified theory is a supermodel of the original was provided. Experimental results showed that the supermodel problem exhibits phase transition behavior similar to that found in other satisfiability work. Appeared at *AAAI'98,* July 1998.

23. "Satisfiability Algorithms and Finite Quantification" [Ginsberg and Parkes, 1998]†
    Although problem descriptions including quantifiers are generally far more compact than their ground counterparts, conventional wisdom is that the constant-factor costs of working with quantified axioms precludes their direct use by modern satisfiability engines. It was shown that in fact the reverse is true: working directly with quantified axioms leads to substantial computational savings that can be expected to outweigh constant-factor costs on large problems. Experimental results supporting this conclusion are also presented. Submitted for publication.

24. "Heuristic optimization: A hybrid AI/OR approach" [Clements *et al.*, 1997]
    This paper described a successful collaboration between CIRL and an Operations Research (OR) group at Georgia Tech on a set of large scale manufacturing scheduling problems. The hybrid approach used the strengths of both fields to produce better

solutions than either AI or OR techniques did on their own. An AI solver was used to produce better initial solutions than OR techniques could, and an OR solver was used to improve on the AI generated solutions by recombining them it ways the AI solver is unable to do. In addition to discussing the collaboration, this paper also introduced Squeaky Wheel Optimization, a new nonsystematic search technique used in the AI solver. Presented at The Workshop on Industrial Constraint-Directed Scheduling at *CP97.*

25. "Squeaky Wheel Optimization" [Joslin and Clements, 1998]
Squeaky wheel optimization is an iterative, nonsystematic, general purpose search technique that is particularly suited to real world problems with complex objective functions. Squeaky wheel uses a dual search space and large coherent moves to avoid many of the problems that local search algorithms suffer from. Every iteration of squeaky wheel moves through two spaces: the traditional solution space, containing all possible solutions; and a new priority space, containing all possible orderings of the elements of the problem. Squeaky wheel navigates between these two spaces using a construct-analyze-prioritize loop. Given some ordering of problem elements (e.g. tasks to schedule), a greedy algorithm is used to construct a solution which is then analyzed to find the elements that are not handled as well as they could be. The results of that analysis are then used to generate new priorities that determine the order in which the greedy algorithm constructs the next solution. All moves in squeaky wheel are made in the priority space, and all moves are attempts to address problems in the previous solution. A single move in the priority space corresponds to a large move in the solution space. Presented at *AAAI-98.*

26. "Cyclic scheduling" [Draper *et al.*, 1998]
This paper considered the problem of cyclic schedules such as arise in manufacturing. A new formulation of this problem was introduced that is a very simple modification of a standard job-shop scheduling formulation, and which enables the use of existing search techniques. Application of this formulation to the more general resource constrained project scheduling (RCPS) problem was also described. Submitted for publication.

9 References

[Bayardo and Schrag, 1997] Roberto J. Bayardo and Robert C. Schrag. Using CSP look-back techniques to solve real-world SAT instances. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, pages 203–208, 1997.

‡[Bedrax-Weiss *et al.*, 1996] Tania Bedrax-Weiss†, Ari Jónsson, and Matthew Ginsberg†. Unsolved problems in planning as constraint satisfaction. Technical report, CIRL, University of Oregon, 1996. CIRL technical report.

‡[Bedrax-Weiss, 1999] Tania Bedrax-Weiss†. *Weighted Discrepancy Search.* PhD thesis, University of Oregon, 1999. in preparation.

[Blum and Furst, 1995] Avrim L. Blum and Merrick L. Furst. Fast planning through planning graph analysis. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 1636–1642, 1995.

‡[Clements *et al.*, 1997] D. Clements†, J. Crawford, D. Joslin, G. Nemhauser, M. Puttlitz, and M. Savelsbergh. Heuristic optimization: A hybrid AI/OR approach. In *Proceedings of the Workshop on Industrial Constraint-Directed Scheduling*, Schloss Hagenberg, Austria, 1997.

[CPLEX Optimization, Inc., 1996] CPLEX Optimization, Inc. Using the CPLEX callable library and CPLEX mixed integer library, Version 4.0. Technical report, CPLEX Optimization, Inc., 1996.

‡[Crawford and Auton, 1996] James M. Crawford and Larry D. Auton. Experimental results on the crossover point in random 3SAT. *Artificial Intelligence*, 81:31–57, 1996.

‡[Crawford and Etherington, 1998] James M. Crawford and David W. Etherington†. A non-deterministic semantics for tractable inference. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, 1998.

‡[Crawford *et al.*, 1996] James Crawford, Matthew Ginsberg†, Eugene Luks, and Amitabha Roy. Symmetry breaking predicates for search problems. In *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning*, Boston, MA, 1996.

‡[Crawford, 1996] James M. Crawford. An approach to resource constrained project scheduling. In *Artificial Intelligence and Manufacturing Research Workshop*, 1996.

[Currie and Tate, 1991] Ken Currie and Austin Tate. O-plan: The open planning architecture. *Artificial Intelligence*, 52:49–86, 1991.

†At CIRL.

‡Work done at CIRL.

‡[Drabble and Clements, 1997] Brian Drabble†and Dave Clements†. Makespan scheduling for assembly tasks: Extended abstract. In *The JPL Workshop on Planning and Scheduling Systems for Space*, October 1997.

[Drabble *et al.*, 1997] Brian Drabble†, Jeff Dalton, and Austin Tate. Repairing plans on the fly. In *The JPL Workshop on Planning and Scheduling Systems for Space*, October 1997.

‡[Drabble *et al.*, 1998a] Brian Drabble†, Terry Lydiard, and Austin Tate. Modeling and supporting the air campaign planning process. Invited, and under review for *Knowledge Engineering Review.*, 1998.

‡[Drabble *et al.*, 1998b] Brian Drabble†, Terry Lydiard, and Austin Tate. Workflow support in the the air campaign planning process. In *Fourth International Conference on Artificial Intelligence Planning Systems (AIPS '98) Workshop on Collaborative Planning*, 1998.

‡[Drabble, 1998] Brian Drabble†. Modern planning and scheduling technologies. *Computing and Control*, 9(3):123–128, June 1998.

‡[Draper *et al.*, 1998] D. Draper, A. Jónsson, D. Clements†, and D. Joslin. Cyclic scheduling. Technical report, CIRL, University of Oregon, 1998. Unpublished.

‡[Etherington and Crawford, 1996] David W. Etherington†and James M. Crawford. Toward efficient default reasoning. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, 1996.

‡[Etherington, 1997] David W. Etherington†. What does knowledge representation have to say to AI? In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, 1997.

‡[Ginsberg and Parkes, 1998] Matthew L. Ginsberg†and Andrew J. Parkes†. Satisfiability algorithms and finite quantification. in preparation, 1998.

‡[Ginsberg *et al.*, 1998a] Matthew L. Ginsberg†, David W. Etherington†, and Drabble Brian. Can search play a role in practical applications? In *Proceedings of AI Meets the Real World '98*, University of Connecticut, Stamford, CT, September 1998.

‡[Ginsberg *et al.*, 1998b] Matthew L. Ginsberg†, Andrew J. Parkes†, and Amitabha Roy. Supermodels and Robustness. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, pages 334–339, Madison, WI, 1998.

‡[Ginsberg, 1995] Matthew L. Ginsberg†. Approximate planning. *Artificial Intelligence*, 76:89–123, 1995.

‡[Ginsberg, 1996a] Matthew L. Ginsberg†. Do computers need common sense? In *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning*, Boston, MA, 1996.

‡[Ginsberg, 1996b] Matthew L. Ginsberg†. A new algorithm for generative planning. In *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning*, Boston, MA, 1996.

‡[Ginsberg, 1996c] Matthew L. Ginsberg†. Partition search. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, 1996.

[Glover and Laguna, 1997] Fred Glover and M. Laguna. *Tabu Search.* Kluwer, 1997.

‡[Harvey and Ginsberg, 1995] William D. Harvey and Matthew L. Ginsberg†. Limited discrepancy search. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 607–613, 1995.

‡[Jónsson and Ginsberg, 1996] Ari K. Jónsson and Matthew L. Ginsberg†. Efficient reasoning using procedures. In *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning*, Boston, MA, 1996.

‡[Jónsson, 1996] Ari K. Jónsson. The CSPC compiler: User's manual. Technical report, CIRL, University of Oregon, 1996.

‡[Joslin and Clements, 1998] D. Joslin and D. Clements†. Squeaky wheel optimization. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, Madison, WI, 1998.

[Joslin and Pollack, 1995] David Joslin and Martha Pollack. Passive and active decision postponement in plan generation. In *European Workshop on Planning*, 1995.

‡[Joslin and Roy, 1997] David Joslin and Amithaba Roy. Exploiting symmetry in lifted constraint satisfaction problems. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, 1997.

[Kautz and Selman, 1996] Henry Kautz and Bart Selman. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, 1996.

[Kautz *et al.*, 1996] Henry Kautz, David McAllester, and Bart Selman. Encoding plans in propositional logic. In *KR-96*, 1996.

‡[Massey, 1998] Barton Massey†. Propositional STRIPS planning problem reversal. ftp://ftp.cirl.uoregon.edu/pub/users/bart/papers/rev.ps.gz, 1998.

[Nemhauser *et al.*, 1994] George Nemhauser, M. Savelsbergh, and G. Sigismondi. MINTO, A mixed integer optimizer. *Operations Research Letters*, 15:47–58, 1994.

‡[Parkes and Walser, 1996] Andrew J. Parkes†and Joachim P. Walser. Tuning Local Search for Satisfiability Testing. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, pages 356–362, Portland, OR, 1996.

‡[Parkes, 1997] Andrew J. Parkes†. Clustering at the Phase Transition. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, pages 340–345, Providence, RI, 1997.

[Selman *et al.*, 1993] Bart Selman, Henry A. Kautz, and Bram Cohen. Local search strategies for satisfiability testing. In *Proceedings 1993 DIMACS Workshop on Maximum Clique, Graph Coloring, and Satisfiability*, 1993.

[SPAR Working Group, 1996] SPAR Working Group. Shared plan and activity representation (SPAR). http://www.aiai.ed.ac.uk/ãrpi/spar/, 1996.

[Stillman *et al.*, 1996] J. Stillman, R. Arthur, and J. Farley. Temporal reasoning for mixed initiative planning. In A. Tate, editor, *Advanced Planning Technology: Technical Achievements of the ARPA/Rome Laboratory Planning Initiative*, pages 242–249, Menlo Park, CA 94025, USA, 1996. The AAAI Press.

[Syswerda, 1994] Gilbert P. Syswerda. Generation of schedules using a genetic procedure. *U.S. Patent number 5,319,781*, 1994.

[Wilkins, 1988] David E. Wilkins. *Practical Planning: Extending the Classical AI Planning Paradigm.* Morgan Kaufmann, San Mateo, CA, 1988.

Appendix A

A New Algorithm for Generative Planning*

Matthew L. Ginsberg
CIRL
1269 University of Oregon
Eugene, OR 97403-1269

February 4, 1999

Abstract

Existing generative planners have two properties that one would like to avoid if possible. First, they use a single mechanism to solve problems both of action selection and of action sequencing, thereby failing to exploit recent progress on scheduling and satisfiability algorithms. Second, the context in which a subgoal is solved is governed in part by the solutions to other subgoals, as opposed to plans for the subgoals being developed in isolation and then merged to yield a plan for the conjunction. We present a reformulation of the planning problem that appears to avoid these difficulties, describing an algorithm that solves subgoals in isolation and then appeals to a separate NP-complete scheduling test to determine whether the actions that have been selected can be combined in a useful way.

1 INTRODUCTION

One of the insights arising from the theoretical analysis of the complexity of domain-independent planning [1, 4, 14] is that planning, involving both the selection and sequencing of actions, is in general properly more difficult than scheduling, which involves sequencing problems in isolation. My goal in this paper is to introduce a new type of planning algorithm that responds to these results by separating the problem of action selection from that of action sequencing. By doing this, we can expect to incorporate into planners fast existing algorithms for solving NP-complete problems such as scheduling.

We will achieve this computational split by introducing a new planning language $\mathcal{L}$ that mediates between the action selection and action sequencing phases. Our intention is that a planner concerned with action selection in isolation could easily produce a sentence of $\mathcal{L}$ as its output, and that a scheduler could accept such a sentence as its input. In solving the Sussman anomaly, for example (Figure 1), the planner would realize that it needs to move block C off block A before moving block A onto block B, and also that it needs to move block B onto block C. The scheduling algorithm would then identify the ordering needed to actually achieve the goal (namely, that it is necessary to move blocks C, B and A in that order).

*This paper appeared in *Proc. KR'96.*

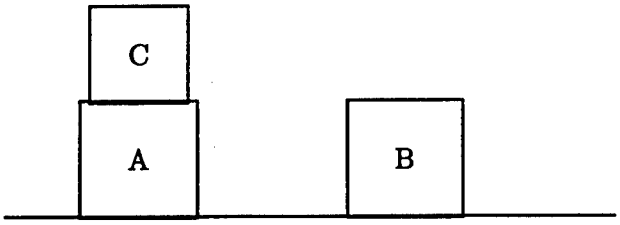


Figure 1: The Sussman anomaly: Get A on B on C

The language itself is introduced in Section 2, and we demonstrate its relevance to scheduling at a theoretical level in Section 3 by showing to be NP-complete the problem of deciding if a particular sentence in the language corresponds to a nonempty set of plans. We turn to planning in Section 4, developing an algorithm that generates instances of our planning language in response to a particular planning problem. This algorithm differs from existing partial-order causal link (POCL) planners in at least three fundamental ways.

First, it solves subgoals separately, merging the results to construct a plan for the overall goal itself. This property is recursive: Subgoals are solved separately at all levels, with the plans being merged to construct one that achieves the overall goal.

Second, it does no backtracking. Instead, the algorithm works by gradually refining a candidate set of plans that might achieve the goal. If a particular approach is tried for a goal g, and this approach involves developing plans to solve a subgoal g', any solutions found for the subgoal will remain available to the planner even if a different approach is eventually selected for g itself.

Finally, the planner needs to plan for distinct goals only once. Even if a particular goal g must be achieved as a precondition for each of two separate actions, there will typically be only a single appearance of g in the planning search space.

We present an example in Section 5, where we also discuss the general computational properties of our procedure. Section 6 discusses an implementation of our work, and Section 7 contains a variety of concluding remarks, connecting our work to a variety of results that have appeared elsewhere. Proofs appear in an appendix.

2 THE PLAN LANGUAGE

We assume throughout that we have some fixed set of actions A.

Definition 2.1 *By a* linear plan *we will mean a finite sequence of elements of A. Such sequences will typically be denoted $\langle a_1, \ldots, a_n \rangle$.*

Of greater interest to us than specific linear plans will be *sets* of such plans. To this end, we define a language describing those sets in which we will be interested:

Definition 2.2 *The* plan language *$\mathcal{L}$ is the smallest set of sentences that contains the distinguished elements $\emptyset$ and $[\,]$ along with every element of A, and is closed under three binary operators $\cup$, $|$ and π.*

The set of plans $\emptyset$ is, as usual, the empty set. The plan $[\,]$ is intended to correspond to the sequence of no actions, and $a \in A \subseteq \mathcal{L}$ is the singleton plan where action a alone is taken. The operator $P|Q$ concatenates the plan Q to the end of the plan P, and $\pi(P,Q)$ is intended to capture the frame axiom in our setting. Let us give the formal definition of π first and then explain its meaning:

Definition 2.3 *Let $P \in \mathcal{L}$ be a set of plans. Membership in P is defined recursively as follows:*

1. $\langle a_1, \ldots, a_n \rangle \notin \emptyset$.
2. $\langle a_1, \ldots, a_n \rangle \in [\,]$ *if and only if* $n = 0$.
3. $\langle a_1, \ldots, a_n \rangle \in a$ *for an action* a *if and only if* $n = 1$ *and* $a_1 = a$.
4. $\langle a_1, \ldots, a_n \rangle \in P|Q$ *if and only if for some* $i \leq n$, $\langle a_1, \ldots, a_i \rangle \in P$ *and* $\langle a_{i+1}, \ldots, a_n \rangle \in Q$.
5. $\langle a_1, \ldots, a_n \rangle \in P \cup Q$ *if and only if* $\langle a_1, \ldots, a_n \rangle \in P$ *or* $\langle a_1, \ldots, a_n \rangle \in Q$.
6. *To determine if* $\langle a_1, \ldots, a_n \rangle \in \pi(P,Q)$, *let* i *be the greatest integer such that* $\langle a_1, \ldots, a_i \rangle \in P \cup Q$. *Then* $\langle a_1, \ldots, a_n \rangle \in \pi(P,Q)$ *if and only if such an* i *exists and* $\langle a_1, \ldots, a_i \rangle \notin Q$.

It is presumably only the last of the above clauses that requires explanation. To understand it, suppose that we have some goal that is established by the plans in P, but disestablished by the plans in Q (clobbered, in Chapman's more picturesque terms [2]). Now the goal will hold after an action sequence $\langle a_1, \ldots, a_n \rangle$ has been executed if and only if it is established by some initial subsequence of $\langle a_1, \ldots, a_n \rangle$ and not subsequently clobbered – in other words, if the longest subsequence $\langle a_1, \ldots, a_i \rangle$ that either establishes or disestablishes the goal in fact establishes it. By writing the definition to require that the action a_i not disestablish the goal, we cater to the possibility that $P \cap Q \neq \emptyset$.

Lemma 2.4 $\pi(P_1, Q_1) \subseteq \pi(P_2, Q_2)$ *if* $P_1 \subseteq P_2$ *and* $Q_1 \supseteq Q_2$. ■

We now make the following notational definition:

Definition 2.5

1. $U \equiv \pi([\,], \emptyset)$.
2. $\neg P \equiv \pi(U, P)$.
3. $P \cap Q \equiv \neg(\neg P \cup \neg Q)$.

Lemma 2.6 *U is the set of all linear plans. Negation and intersection have their conventional meanings.*

In the Sussman anomaly, for example, the goal of getting A on B is established by moving A to B, which succeeds in at most the plans given by $U|\mathtt{move}(A,B)$. Moving C to B succeeds for at most the plans in $U|\mathtt{move}(C,B)$. For the preconditions of the various actions, we define

$$\mathtt{cover}(B) \equiv [U|\mathtt{move}(A,B)] \cup [U|\mathtt{move}(C,B)] \tag{1}$$

the set of all plans that *might* clobber B's being clear, and similarly for $\mathtt{cover}(A)$ and $\mathtt{cover}(C)$. We can now apply Lemma 2.4 to conclude that since B is clear in the initial situation, it will remain clear after executing at *least* the plans in the set

$$\mathtt{clear}(B) \equiv \pi([\,], \mathtt{cover}(B)) \tag{2}$$

since we have perhaps overestimated the set of clobberers in (1). The above set is the set of all action sequences that do not contain either $\mathtt{move}(A,B)$ or $\mathtt{move}(C,B)$, and B will be clear after any such sequence is executed, since it is clear initially.

In a similar way, C is clear for at least the plans in the set $\mathtt{clear}(C) \equiv \pi([\,], \mathtt{cover}(C))$ and C is on A for at least

$$\mathtt{on}(C,A) \equiv \pi([\,], U|\mathtt{move}(C,B) \cup U|\mathtt{move}(C,\mathtt{table}))$$

It follows that A will be made clear by the plans in the set

$$[\mathtt{clear}(C) \cap \mathtt{on}(C,A)]|\mathtt{move}(C,\mathtt{table})$$

and therefore that A will be clear for plans in the set

$$\mathtt{clear}(A) \equiv$$
$$\pi([\mathtt{clear}(C) \cap \mathtt{on}(C,A)]|\mathtt{move}(C,\mathtt{table}), \mathtt{cover}(A))$$

If C is clear and on A, A will be clear in a persistent way after C is moved to the table. Covering A (the second argument to π above) defeats the persistence.

We can now conclude that A will be on B for at least the plans in the set

$$\mathtt{on}(A,B) \equiv \pi([\mathtt{clear}(A) \cap \mathtt{clear}(B)]|\mathtt{move}(A,B),$$
$$[U|\mathtt{move}(A,C)] \cup [U|\mathtt{move}(A,\mathtt{table})])$$

In a similar way, B will be on C for at least the plans

$$\mathtt{on}(B,C) \equiv \pi([\mathtt{clear}(B) \cap \mathtt{clear}(C)]|\mathtt{move}(B,C),$$
$$[U|\mathtt{move}(B,A)] \cup [U|\mathtt{move}(B,\mathtt{table})])$$

The overall goal will be achieved by plans in the set

$$\mathtt{on}(A,B) \cap \mathtt{on}(B,C) \tag{3}$$

Consider now the successful plan

$$\langle \mathtt{move}(C,\mathtt{table}), \mathtt{move}(B,C), \mathtt{move}(A,B) \rangle \tag{4}$$

This plan is in $\mathtt{on}(A, B)$ because the plan itself is in $[\mathtt{clear}(A) \cap \mathtt{clear}(B)]|\mathtt{move}(A, B)$. To see this, we must show that

$$\langle \mathtt{move}(C, \mathtt{table}), \mathtt{move}(B, C) \rangle \tag{5}$$

is in $\mathtt{clear}(A) \cap \mathtt{clear}(B)$. To see *this*, note that no subsequence of (5) is in $\mathtt{cover}(A)$ or $\mathtt{cover}(B)$. As a result, we can show that (5) is in $\mathtt{clear}(A)$ by showing that some subsequence of (5) is in

$$[\mathtt{clear}(C) \cap \mathtt{on}(C, A)]|\mathtt{move}(C, \mathtt{table})$$

In a similar way, we can show that (5) is in $\mathtt{clear}(B)$ by showing that some subsequence of (5) is in $[\,]$.

The second of these is obvious. The first follows because $[\,]$ is in $\mathtt{clear}(C) \cap \mathtt{on}(C, A)$. As a result, we can conclude that A and B are clear after (5), so that A is indeed on B after executing (4). It is not hard to show that B is on C as well, so that (4) does indeed succeed in solving our problem.

What about the failing plan

$$\langle \mathtt{move}(C, \mathtt{table}), \mathtt{move}(A, B), \mathtt{move}(B, C) \rangle \tag{6}$$

where A is moved onto B too early? In this case, we can show that (6) is not known to be in $\mathtt{on}(B, C)$. Specifically, it has no subsequence known to be in

$$[\mathtt{clear}(B) \cap \mathtt{clear}(C)]|\mathtt{move}(B, C)$$

because $\langle \mathtt{move}(C, \mathtt{table}), \mathtt{move}(A, B) \rangle$ is not known to be in $\mathtt{clear}(B) \cap \mathtt{clear}(C)$. Still more specifically, $\langle \mathtt{move}(C, \mathtt{table}), \mathtt{move}(A, B) \rangle$ is not in $\mathtt{clear}(B)$ because the sequence itself is in $\mathtt{cover}(B)$.

3 COMPLEXITY RESULTS

The analysis of the previous section shows that the successful plan

$$\langle \mathtt{move}(C, \mathtt{table}), \mathtt{move}(B, C), \mathtt{move}(A, B) \rangle$$

is a member of (3) while the failing one

$$\langle \mathtt{move}(C, \mathtt{table}), \mathtt{move}(A, B), \mathtt{move}(B, C) \rangle$$

is not. In some sense, this means that membership in (3) resolves the question of whether a specific linear plan satisfies the ordering constraints implicit in the original problem. Our main goal in this section will be to show the following:

Proposition 3.1 *Let $P \in \mathcal{L}$ be a set of plans. Then showing that $P \neq \emptyset$ is NP-complete.*

The proof involves two stages. First, we must show that the problem in question is in NP by demonstrating that it is possible to validate in polynomial time a witness that $P \neq \emptyset$. Second, we need to show that the problem is NP-hard by embedding some known NP-hard problem in it. For the first part, we have:

Definition 3.2 *For any $P \in \mathcal{L}$, we define the* length *of P to be the length of the syntactic expression P itself. The length of P will be denoted $|P|$.*

Lemma 3.3 *Let $P \in \mathcal{L}$ be a nonempty set of plans. Then there is a plan $\langle a_1, \ldots, a_n \rangle \in P$ with $n \leq 1 + |P|$.*

Lemma 3.4 *Let $P \in \mathcal{L}$ be a set of plans, and p a specific linear plan. Then the question of whether $p \in P$ can be resolved in time polynomial in the lengths of p and P.*

Lemma 3.4 tells us that we can validate a witness that $P \neq \emptyset$ in time polynomial in both the size of P and the length of the witness; Lemma 3.3 guarantees that the witness itself is of polynomial size in P.

Corollary 3.5 *Let $P \in \mathcal{L}$ be a set of plans. Showing that $P \neq \emptyset$ is in NP.* ■

To show that the problem is NP-hard, consider the known NP-complete problem of satisfiability. In this problem, we have a set of clauses such as $x_1 \vee \neg x_2 \vee x_3$ and need to find an assignment of true or false to each of the x_i's such that every clause is satisfied. To achieve the embedding, we will view the x_i's as actions as well, taking x_i to mean that x_i is an action in some hypothetical plan p. Now the clauses can all be satisfied if and only if a plan can be found meeting the given conditions.

Definition 3.6 *Let X be a set of atomic variables. Now if $x \in X$ is an atom, we define $p(x)$ to be $\pi(U|x, \emptyset)$. For a literal that is the negation of an atom, we take $p(\neg x) = \neg p(x)$. If $c = \bigvee_i l_i$ is a clause (a disjunction of literals), we take $p(c) = \bigcup_i p(l_i)$, and if $C = \bigwedge_i c_i$ is a theory (a conjunction of clauses), we take $p(C) = \bigcap_i p(c_i)$.*

Lemma 3.7 *A theory C is satisfiable if and only if $p(C) \neq \emptyset$.*

Corollary 3.8 *Let $P \in \mathcal{L}$ be a set of plans. Showing that $P \neq \emptyset$ is NP-hard.* ■

Proposition 3.1 now follows.

4 PLANNING

In this section, we present an algorithm that works from a domain description to compute the set of plans that achieve a particular goal. We will assume that the domain is described in terms of STRIPS-like add, delete, and precondition lists, but this assumption is not central to our methods. We will also assume the existence of an "initializing" action i that sets up

the initial state. This action has no preconditions, adds all fluents true in the initial state, and deletes all others.

Let us begin with some fairly high-level remarks. In any realistic situation, it is unlikely that we be able to compute exactly the set of plans achieving a goal g. The axiomatization of the domain can be expected to include information implying the existence of a wide range of exceptions to any specific plan or plan schema that achieves g. These exceptions will themselves have exceptions, plans that achieve g even though they belong to "schema" that are expected not to, and so on. If we denote by $L(g)$ the set of plans that achieve g, we see that it will in general be impractical to attempt to compute $L(g)$ exactly.

Instead, we will give a procedure that computes a set of increasingly fine approximations to $L(g)$. Somewhat more specifically, we will produce sequences of plans

$$L_0^+(g) \supseteq L_1^+(g) \supseteq \cdots$$

and

$$L_0^-(g) \subseteq L_1^-(g) \subseteq \cdots$$

such that

$$L_0^-(g) \subseteq L_1^-(g) \subseteq \ldots \subseteq L(g) \subseteq \ldots \subseteq L_1^+(g) \subseteq L_0^+(g)$$

In other words, $L(g)$ is always bracketed by the L^- and L^+ approximations. Since $L(g) \supseteq L_k^-(g)$ for any k, we can stop and return *some* successful plan as soon as $L_k^-(g) \neq \emptyset$. (Note that the check that $L_k^-(g) \neq \emptyset$ is NP-complete.) Alternatively, we might stop and return when L^+ and L^- are "close" in some sense, as is suggested in the work on approximate planning [9].

My interest here, however, is simply in constructing the approximations L^+ and L^- themselves. To that end, we will think of a node in the planning search tree as corresponding either to a goal or subgoal (a **goal** node) or to a specific action that we intend to take (an **action** node). In addition, we will label nodes with a *parity* in the sense that "positive" nodes support plans to achieve the goal and "negative" nodes correspond to exceptions to those plans or other difficulties. A node can also be *expanded* or *unexpanded.*

Definition 4.1 *A* planning tree *is a singly rooted, directed acyclic graph where no node has paths back to the root of both even and odd length. Nodes at even depths (including the root) will be called* goal *nodes and denoted* $(g, \pm)$*; nodes at odd depths will be called* action *nodes and denoted* $(a, \pm)$*. Each fringe node in the graph can be* expanded *or* unexpanded.

If T *is a planning tree and* n *is an unexpanded fringe node in* T*, the* result of expanding n in T *is the planning tree obtained by labeling* n *as expanded and adding as children of* n *the following nodes:*

1. If $n = (g, \pm)$ *is a goal node, the children are action nodes* $(a, \pm)$ *for each action that adds* g *and action nodes* $(a, \mp)$ *for each action that deletes* g*.*

2. If $n = (a, \pm)$ *is an action node, the children are goal nodes* $(g, \pm)$ *for each precondition* g *of* a*.*

A legal tree *for a goal* g *is any tree that can be obtained by successively expanding the tree containing the single node* $(g, +)$*.*

Of course, it is not sufficient to simply build the trees; we also need to label the nodes with the plans that achieve them. To this end, we define:

Definition 4.2 *In a* conservative *labeling, every unexpanded node in a planning tree with positive parity is labeled $\emptyset$ and every unexpanded node with negative parity is labeled U. In a* liberal *labeling, every unexpanded node with positive parity is labeled U and every unexpanded node with negative parity is labeled $\emptyset$.*

The label for any expanded action node (a,p) is

$$\bigcap_i l(c_i)|a \tag{7}$$

where the c_i are the children of (a,p) and the labels $l(c_i)$ are the labels attached to those children. Any expanded goal node (g,p) is labeled with

$$\pi\left(\bigcup_i l(c_{i,p}), \bigcup_i l(c_{i,\neg p})\right)$$

where the $c_{i,p}$ are the children of the same parity as (g,p) and the $c_{i,\neg p}$ are the children of opposite parity.

The conservative *(respectively* liberal*) labeling of a tree T is the label assigned to the root node by a conservative (respectively liberal) labeling of T. These labelings are denoted $L^-(T)$ and $L^+(T)$ respectively.*

Theorem 4.3 *Let T_0 be a planning tree, and T_1 an expansion of it. Suppose that T_0 is labeled L_0^+ in a liberal labeling and L_0^- in a conservative one, and that T_1 is labeled L_1^+ and L_1^-. Then*

$$L_0^- \subseteq L_1^- \subseteq L(g) \subseteq L_1^+ \subseteq L_0^+$$

where g is the goal at the root of T_0.

The above theorem in isolation tells us little; it would be satisfied if $L_k^- = \emptyset$ and $L_k^+ = U$ for all k! We still need to know that the planning algorithm we have proposed actually converges on the correct answer:

Theorem 4.4 *Let g be a goal, and T_0 the planning tree consisting of the single goal node $(g,+)$. Now if p is a plan that achieves g, there is some expansion T of T_0 such that $p \in L^-(T)$. Conversely, if p fails to achieve g, there is an expansion such that $p \notin L^+(T)$.*

5 COMPUTATION

The results of the previous section sanction the following planning algorithm:

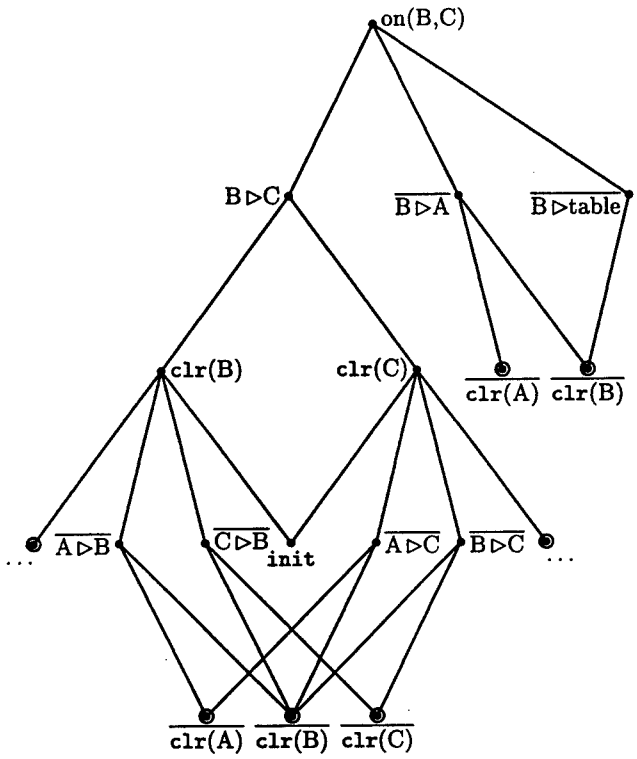


Figure 2: Planning to get B on C

Procedure 5.1 To find a plan for a goal g:

$T :=$ the planning tree consisting of the single
    node $(g, +)$
while $L^-(T) = \emptyset$
    if T contains an unexpanded node, expand it
    else fail
return $L^-(T)$

Theorem 5.2 *If Procedure 5.1 returns a set of plans P in response to a goal g, every plan in P achieves g. In addition, if the nodes are expanded in order of increasing depth, the procedure will always terminate for any achievable goal g.* ■

Procedure 5.1 can be modified to check at each step whether $L^+(T) = \emptyset$. Should this ever happen, it is possible for the planner to report failure even if the tree has not yet been fully expanded.

As an example of the procedure at work, consider the simple goal of getting B onto C in the Sussman anomaly. A partial expansion of the planning tree for this problem appears in Figure 2. The nodes in the figure are labeled as follows:

1. Action nodes are labeled with the action in question; the action of moving A to B, for example, is denoted $A \rhd B$. The initializing action is denoted `init`.

2. Goal nodes are labeled with the fluent that the node is trying to achieve. A node labeled $\texttt{clr}(B)$, for example, is trying to get B clear.

3. Nodes of negative parity appear with an overline. Nodes of positive parity are not so marked.

4. Unexpanded nodes are marked with a bullseye.

The reasoning behind the arcs in the graph is as follows:

1. The root node is labeled with the goal $\texttt{on}(B, C)$.

2. The root can be achieved by moving B to C. It can be clobbered by moving B to A or to the table. These three children are added at depth 1, with the nodes corresponding to the clobbering actions having negative parity.

3. Moving B to C requires that B and C be clear; moving B to A (a clobbering action) requires that B and A be clear. Since the two nodes at depth two with goal $\texttt{clear}(B)$ have opposing parities, they cannot be combined in the planning tree. The two appearances of $\texttt{clear}(B)$ under the clobbering actions *are* combined.

4. B is made clear by the initializing action, and also potentially by a variety of other actions that have been lumped into a single node labeled ... in the figure. (Moving A off the top of B will make B clear, and so on.) B being clear will be clobbered by moving either A or C onto B. The goal node labeled $\texttt{clr}(C)$ is expanded similarly.

5. The negative parity action node $A \rhd B$ at depth 3 has as children negative parity goal nodes trying to get A and B clear. These nodes could be identified with analogous nodes at depth 2, but haven't been in the interests of maintaining the clarity of the figure.

What about the labels assigned to the nodes? Assuming that we are interested in a conservative labeling of the tree, we begin by assigning either $\emptyset$ or U to the unexpanded nodes depending on their parity. The fringe nodes at the bottom of the tree (depth 4) and those at depth 2 are all of negative parity and are labeled U, while the unexpanded ... nodes at depth 3 are labeled $\emptyset$.

We can now compute the labels for the nodes at depth 3. These labels include expressions of the form $U|\texttt{init}$, indicating that the initial situation needs to be established before the plans involved can be effective. We abbreviate $U|\texttt{init}$ to $[\,]$ to get the tree in Figure 3. In the figure, we have dropped nodes labeled $\emptyset$ that cannot contribute to the values of nodes at depth 2, and have abbreviated $U|\texttt{move}(A, B)$ (for example) to $U|\mathrm{A} \rhd \mathrm{B}$.

Consider now the node at depth 2 in Figure 3 dealing with the goal $\texttt{clear}(B)$. The label to be assigned to this node is

$$\pi([\,], U|\texttt{move}(A, B) \cup U|\texttt{move}(C, B))$$

which is exactly the set of plans that we denoted $\texttt{clear}(B)$ in (2) of Section 2.

Continuing, the label assigned to the action node $B \rhd C$ in the figure is seen to be

$$[\texttt{clear}(B) \cap \texttt{clear}(C)]|\texttt{move}(B, C) \tag{8}$$

and the labels assigned to the two action nodes of negative parity are $U|\texttt{move}(B, A)$ and $[U|\texttt{move}(B, \texttt{table})$ respectively. We can finally label the root node as

$$\pi([\texttt{clear}(B) \cap \texttt{clear}(C)]|\texttt{move}(B, C),$$

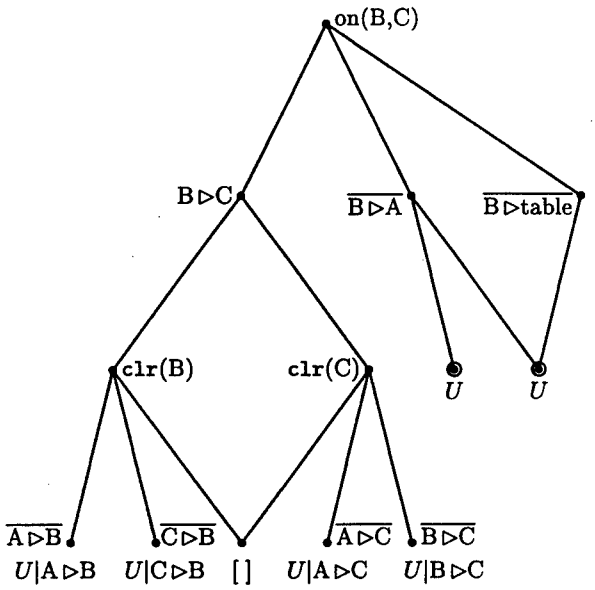


Figure 3: The conservative labeling, continued

$$[U|\texttt{move}(B,A)] \cup [U|\texttt{move}(B,\texttt{table})]) \tag{9}$$

This plan set is nonempty (the sequence $\langle \texttt{move}(B,C) \rangle$ is an element of it), and the planner can return at this point.

Note that the planner cannot return until the two negative parity nodes at depth 1 have been expanded. If these were not expanded, (9) would be replaced with

$$\pi([\texttt{clear}(B) \cap \texttt{clear}(C)]|\texttt{move}(B,C),U)$$

which is empty.

After the negative parity nodes at depth two are expanded, we see that specific actions are needed to clobber the goal $\texttt{on}(B,C)$; if these actions are not taken, the goal is achieved. The plan set (9) is nonempty because it is indeed possible to instantiate the plan without clobbering the overall goal.

Note also that although we chose to fully expand the planning tree in this example, there was no intrinsic need for us to do so. If we had identified only some of the actions that might achieve or clobber a particular goal, we would still have been able to construct plan sets in a similar fashion, although the bounds we obtained on the set of plans that actually *do* achieve our goal would necessarily have been somewhat looser.

If we had been working with some other (and presumably more sophisticated) semantics of action, our overall approach would have been unchanged. It would still have been possible for us to construct a tree describing the actions that might achieve or clobber the various goals that were of interest to us, and we would be able to employ quite similar techniques to construct and to then evaluate a sentence in the planning language.

Finally, we remarked in the introduction that unlike conventional planners (POCL or otherwise), Procedure 5.1 plans for subgoals separately. What this means is that once a

plan is found for a subgoal anywhere in the tree that is "successful" in that it can be incorporated into a successful plan for the goal at the root, the node corresponding to the subgoal need not be revisited.

Formalizing this is clearest if we assume that nodes with identical parities and labels have not been identified (a conventional planner would be incapable of identifying them in any event). As a preliminary, we make the following definition:

Definition 5.3 *Suppose that T_1 and T_2 are two planning trees with identical roots, and that both are trees as opposed to directed acyclic graphs. We will say that T_1 and T_2* match *at a node n appearing in each of them if (1) the path from n back to the root is the same in the two trees, and (2) the expansions of the two trees under n are identical.*

Proposition 5.4 *Suppose that T is a partially expanded planning tree for a goal g, and that n is a positive node in T. Now if there is any planning tree T' for g that matches T at n and for which $L^-(T') \neq \emptyset$, there is an expansion T_e of T with $L^-(T_e) \neq \emptyset$ such that T_e and T match at n.*

In other words, once we have found an expansion of a node n that participates in any solution of the original planning problem (as witnessed by the tree T'), we need never examine a node below n again.

6 IMPLEMENTATION

We have built a prototype implementation of our ideas that conforms to the theoretical description that we have presented. The planner is called COPS (COmbined Planner and Scheduler).

COPS accepts as input a goal and a set of action descriptors. Each descriptor includes the name of the action, its preconditions, add and delete lists, and a list of resources used by the action. COPS begins by building a planning tree using the mechanisms we have described, ignoring the resource information as it does so. The user can have the system construct the tree in its entirety, or can control the node expansions so that only the portion of interest is built.

Either after the planning tree is complete or while it is being constructed, the user can select any node in the tree for either liberal or conservative evaluation. A sentence in the planning language is then constructed and translated into an equivalent sentence in a finite, sorted first-order logic. This translation incorporates additional constraints that handle potential conflicts due to resource over-utilization by the various actions.

The first-order theory is converted into a (once again equivalent) predicate theory and passed to a predicate logic prover to check for satisfiability. The implementation uses a modification of TABLEAU [3] that incorporates procedural reasoning capabilities [11]. While the original sentence in the planning language directly captures only some of the ordering constraints on the actions involved (and therefore might best be thought of as partial-order), the theory in predicate logic describes conditions that linear plans need to satisfy in order to achieve the overall goals.

The performance of the system is as one would expect. Interactions corresponding to single actions establishing or disestablishing multiple subgoals correspond to entanglements in the planning tree much like those present in Figure 2; these entanglements are eventually resolved by the predicate prover. Interactions that are a consequence of resource contentions are handled by the predicate prover in isolation, with the structure of the planning tree remaining simple in this case.

In the future, we will work to extend COPS in a variety of ways. First, we will exploit the similarity between the planner's existing architecture and the theorem proving system MVL [7] so that actions can be described using declarative methods instead of a STRIPS-like representation. This will also allow us to treat actions with variables in their description; the current implementation is limited to the ground case.

We will also work to apply COPS to a variety of different domains, and to evaluate its scaling behavior in each. In cases where subgoal interactions correspond to resource contentions, we expect the system's scaling properties to match those of the underlying predicate prover/scheduler; in cases where the interactions correspond to multiple uses of single actions, COPS will doubtless suffer from combinatorial difficulties similar to those encountered by conventional methods. We expect these difficulties to be mediated by the facts that some of the complexity will be deferred to the first-order prover and COPS can recognize repeated subgoals independent of the contexts in which they appear.

7 CONCLUDING REMARKS

The COPS algorithm is only a beginning; there is, for example, no guidance with regard to the selection of the planning node to be expanded next. In this section, we draw connections between our ideas and others that may shed light on this or other issues.

7.1 EXISTING PLANNING WORK

We begin by briefly comparing our ideas and two existing planners, Joslin's DESCARTES [12] and Etzioni's STATIC [5, 6]. DESCARTES is in the POCL class, while STATIC is not.

The DESCARTES planner works to identify the constraint-satisfaction problems corresponding to action scheduling and to then schedule the actions using a dedicated constraint-satisfaction or scheduling algorithm of some sort [12]; by taking this approach, DESCARTES attempts to use the search facilities of the scheduling engine to minimize the search undertaken by the planner per se. This idea is similar to ours, although DESCARTES is capable of making tentative and unforced commitments when it appears that the associated constraint-satisfaction problem has become underconstrained. These "early commitments" appear to have no counterpart in our method.

In the absence of early commitments, the behavior of the two systems is strikingly similar. Both use plans for achieving single subgoals in a variety of contexts, and both suffer from the potential difficulty that problems repeatedly passed to the scheduler share many features with their predecessors. In DESCARTES's case, it would be useful if the results of prior scheduling searches could inform subsequent ones; in our case, the determination of

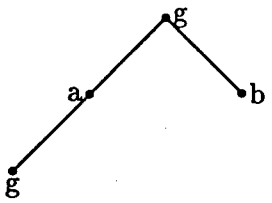


Figure 4: A fortuitous recursion

whether $L^-(T) = \emptyset$ can be expected to be similar from one expansion of the planning tree to the next and it will clearly be important to exploit these similarities computationally.

The other connection we would like to draw is between our work and other attempts to combine identical subgoals in planning. At an intuitive level, the development of macro operators [13] is an attempt in this direction. By identifying specific sequences of actions that achieve common goals, the cost of planning can be reduced. Our approach realizes similar advantages if a single sequence of actions can be used to achieve a repeated goal multiple times and can also deal with situations where differing sequences of actions are needed. In this latter case, however, there is some computational cost to our methods, since both sequences will need to be evaluated in the check to see whether $L^-(g) = \emptyset$.

Another author who has attempted to identify recurring subgoals is Etzioni [5]. Etzioni's "problem space graphs" (PSG's) are strikingly similar to our planning trees, consisting as they do of "alternating sequence[s] of subgoals and operators" [5, page 262].[1] The PSG's are not used to solve any planning problem specifically, but instead capture the dependencies between goals and subgoals that are part of the overall problem formulation.

Etzioni describes as "fortuitous" recursions that are of a structural form that can be identified by an analysis of the PSG for the domain in question, and he shows that the blocks world is indeed fortuitous. Unfortunately, no criteria are presented whereby fortuitous recursions can be identified analytically.

We can do a bit better than this. If a node in a planning tree is identical to one of its ancestors, we can denote the label assigned to the node by $L(n)$ and then produce a fixed-point equation that $L(n)$ needs to satisfy. The actual value for $L(n)$ will be the smallest set of plans satisfying the equation. In many cases, it may be possible to show that the label for the ancestor node is in fact independent of the label for the child, in which case we will have shown that the recursion is fortuitous and the child can be pruned.

An example of this phenomenon appears in Figure 4. The goal g has no clobberers and can be achieved by either a or b. The preconditions for a are g, while b has no preconditions.

Suppose that we denote the eventual label for g by L. Now the label for a is $L|a$, while

[1] Smith and Peot's *operator graphs* [15] are also similar, as Dan Weld has pointed out to me. The principal differences between PSG/operator graphs and our approach are that (1) PSG's (but not operator graphs) are constructed statically from the domain description, and not in response to a particular problem instance, and (2) PSG's and operator graphs both include information about goal establishment only, and ignore deletions and disestablishment.

the label for b is $U|b$. The label for g at the root of the tree is therefore

$$\pi(L|a \cup U|b, \emptyset) = L \tag{10}$$

and it is not hard to see that the least solution to (10) is $L = \pi(U|b, \emptyset)$; g is achieved if we ever execute the action b, but in no other cases. In this case, we see that the repeated subgoal g can be pruned without affecting the label of the original. In fact, we can show:

Proposition 7.1 *Any subgoal with no clobberers in its subtree is fortuitous.*

7.2 BILATTICES AND MODALITY

The planning tree structure that we have introduced is similar in many ways to existing work on truth-functional interpretations of modality [8, 10]. Underlying this interpretation is the notion that sentences should be labeled not merely as "true" or "false" but instead with a richer set of truth values selected from a *bilattice* [7]. A bilattice is a set equipped with two partial orders, one indicating how true or false a particular sentence is, and the other partial order indicating how complete one's state of knowledge is.

In this setting, it is possible to respond to a modal query like, "I know it is raining outside," by invoking a theorem prover recursively on the embedded sentential argument ("It is raining outside") and then manipulating the truth value returned as a result [8]. One of the interesting features of naturally occurring modalities is that they tend to either preserve or to invert the "truth" partial order. As an example, "true" (t) is more true than "unknown" (u). The modal operator of knowledge (or necessity), applied to t, returns t. Applied to u, it returns f (false). We might write $K(t) = t$ and $K(u) = f$.

With regard to the truth partial order, $t >_t u$ and $K(t) >_t K(u)$. The "knowledge" partial order is not preserved; $t >_k u$ but $K(t)$ and $K(u)$ are incomparable because neither corresponds to a more complete state of knowledge than the other. It is not clear why natural modalities should be either t-monotonic or t-antimonotonic in this way.

Now consider our planning language. A set of plans P can be thought of as a function ϕ from the set U of all plans into the two-point set $\{t, f\}$. A specific plan p is in P if and only if $\phi(p) = t$. The set of such functions can be interpreted as a bilattice by first embedding the set $\{t, f\}$ in the four element bilattice $F = \{t, f, u, \perp\}$ and then realizing that F^U, the set of functions from U to F, inherits a bilattice structure from the bilattice structure on F. Suppose that we denote this "planning" bilattice by $\mathcal{P}$.

An operator on sets of plans such as $|$ or π is now nothing more than a function that itself operates on functions from U to $\{t, f\}$. In other words, $|$ and π are binary functions from $\mathcal{P} \times \mathcal{P}$ to $\mathcal{P}$. As such, they can be interpreted as modal operators on the planning bilattice. The bilattice-based modal theorem prover [10] is, in fact, little more than a lifting of Procedure 5.1 to a more general setting. Once again, the modalities preserve or invert the t partial order. This is clear for $|$; increasing the set of plans in P or Q increases the set of plans in $P|Q$. The operator π, on the other hand, is monotonic in its first argument and antimonotonic in its second. Perhaps the planning results will shed some light on the bilattice theory here, since we know from results in this paper that the monotonicity and antimonotonicity underlie the convergence properties of the overall algorithms.

7.3 APPROXIMATE PLANNING

Finally, consider the termination criteria in Procedure 5.1: We stop as soon as $L^-(g) \neq \emptyset$, showing that the set of plans $L(g)$ for achieving the goal g is nonempty. There is an optional check to see if $L^+(g) = \emptyset$, in which case $L(g) = \emptyset$ as well and we can terminate and report that the goal is not achievable.

There are other possibilities also. One is to stop whenever the bracketing sets $L^+(T)$ and $L^-(T)$ are in some sense "close" to one another; this is similar to suggestions made in the work on *approximate planning* [9].

The basic idea here is to define conditions under which one set of plans is "small" relative to another. As an example, the set of plans $\pi(P|a, \emptyset)$ will typically be small in the set $\pi(P, \emptyset)$ since the first set involves a commitment to take the action a while the second set does not.

We can now go on to say that two sets $P \subseteq Q$ are *approximately equal*, writing $P \approx Q$, if their difference $Q - P$ is small relative to Q. Consider, for example, the plan

$$\pi(U|\mathtt{move}(B, C), \emptyset) \tag{11}$$

for getting B onto C in the Sussman anomaly. In order for this plan to fail, one of a handful of specific actions must be taken: Something (A or C) needs to be moved onto B, something needs to be moved onto C, or B needs to be moved away after it is moved onto C successfully. Each of these "exception" plans is small relative to the overall plan schema (11), so (11) itself is approximately equal to the set of plans that actually achieve the goal.

We can modify our basic procedure to return these approximate plans by changing the termination condition from $L^-(T) \neq \emptyset$ to $L^-(T) \approx L^+(T)$. Doing so produces a procedure that implements the approximate planning ideas that have appeared elsewhere, although the complexity of checking whether $P \approx Q$ for $P \subseteq Q$ and $P, Q \in \mathcal{L}$ is not yet known.

All told, the ideas and algorithms that we have described seem to raise as many new questions as they answer. The new questions are, however, quite different from those that have typically been considered by the generative planning community. Perhaps the differences themselves are an indication of progress.

A PROOFS

Lemma 2.6 *U is the set of all linear plans. Negation and intersection have their conventional meanings.*

Proof. To see that $\langle a_1, \ldots, a_n \rangle \in U$ for any plan, note that the longest subsequence of $\langle a_1, \ldots, a_n \rangle$ in $[\,] \cup \emptyset$ is the initial empty subsequence, which is not in $\emptyset$.

For negation, Suppose that $\langle a_1, \ldots, a_n \rangle \in P$. Now it is in $U \cup P$, and also in P, so $\langle a_1, \ldots, a_n \rangle \notin \neg P = \pi(U, P)$. Alternatively, if $\langle a_1, \ldots, a_n \rangle \notin P$, it is in $U \cup P = U$ anyway, so $\langle a_1, \ldots, a_n \rangle \in \neg P$. The result for intersection follows from this. ■

Lemma 3.3 *Let $P \in \mathcal{L}$ be a nonempty set of plans. Then there is a plan $\langle a_1, \ldots, a_n \rangle \in P$ with $n \leq 1 + |P|$.*

Proof. Let $\langle a_1, \ldots, a_k \rangle$ be a linear plan. We will show that there is some subset S of the indices $\{1, \ldots, k\}$ with the size of S at least $k - 1 - |P|$ and such that we can remove any subset of the actions in $\langle a_1, \ldots, a_k \rangle$ whose indices are in S without affecting membership in P. The result follows; given any element of P containing k actions, we can drop at least $k - 1 - |P|$ of them to get an element of P containing at most $1 + |P|$ actions.

The proof is by structural induction on the set of plans P. For any of the base cases $\emptyset$, $[\,]$ or $a \in A$, the result is clear. (At most two actions are needed to resolve the question of membership in P, and $|P| = 1$.)

For the compound cases, consider $P|Q$ first. There is now a set S_p of appearances in P where actions can be dropped, and a set S_q of similar appearances in Q. We take $S = S_p \cap S_q$, so that membership in both P and Q is unaffected. Now the size of S is at least

$$\begin{aligned} |S_p| + |S_q| - k &\geq k - 1 - |P| + k - 1 - |Q| - k \\ &= k - 1 - (|P| + |Q| + 1) \\ &= k - 1 - |(P|Q)| \end{aligned}$$

since the size of $P|Q$ includes an additional count for the concatenation symbol. $\bigcup$ and π can be handled identically. ■

Lemma 3.4 *Let $P \in \mathcal{L}$ be a set of plans, and p a specific linear plan. Then the question of whether $p \in P$ can be resolved in time polynomial in the lengths of p and P.*

Proof. The only subtlety involves the clauses defining $P|Q$ and $\pi(P, Q)$ in Definition 2.3, since these clauses ostensibly involve a search over subsequences of p. We take a dynamic programming approach, investigating all of the subsequences of p simultaneously and working from the component plans of P outward.

Somewhat more specifically, suppose that Q is a set of plans, $p = \langle a_1, \ldots, a_n \rangle$, and for each top level plan Q_i appearing in Q and each $j \leq k \leq n$, we know whether $\langle a_j, \ldots, a_k \rangle \in Q_i$. Now we can determine in time $o(k)$ whether $\langle a_1, \ldots, a_k \rangle \in Q$. If Q is the concatenation of two plans, we consider each point at which the first plan might stop and the second begin. If Q is the result of applying π, we consider each initial subsequence of p starting with the longest (i.e., p itself).

Since there are at most n^2 subsequences of the plan p, it follows that we can determine membership in Q for each of them in time at most n^3 overall. If l is the length of the original set of plans P, it follows that we can determine in time $o(n^3 l)$ whether $\langle a_j, \ldots, a_k \rangle \in P$ for each j and k, and thus whether $p \in P$. ■

Lemma 3.7 *A theory C is satisfiable if and only if $p(C) \neq \emptyset$.*

Proof. It is clear from Definition 3.6 that the result will follow if we can show that for any atom x_j and linear plan $\langle x_1, \ldots, x_n \rangle$, $\langle x_1, \ldots, x_n \rangle \in p(x_j)$ if and only if $x_j \in \{x_i\}$ This is immediate from the definition, however. ■

Theorem 4.3 *Let T_0 be a planning tree, and T_1 an expansion of it. Suppose that the root of T_0 is labeled L_0^+ in a liberal labeling and L_0^- in a conservative one, and that the root of T_1 is labeled L_1^+ and L_1^-. Then*

$$L_0^- \subseteq L_1^- \subseteq L(g) \subseteq L_1^+ \subseteq L_0^+ \tag{12}$$

where g is the goal at the root of T_0.

Proof. The proof consists of two parts. In the first, we show that $L_0^- \subseteq L_1^-$; in the second, that $L_0^- \subseteq L(g)$. Since the argument is independent of the actual tree T_0, it will follow that $L_1^- \subseteq L(g)$ as well. The right hand inclusions in (12) follow using dual arguments.

To see that $L_0^- \subseteq L_1^-$, suppose that T is a planning tree, and T' an expansion of it. Suppose also that n is any node in T. If a conservative labeling of T labels n with P while a conservative labeling of T' labels n with P', we claim that $P \subseteq P'$ if the parity of n is positive, and $P \supseteq P'$ if the parity of n is negative.

We show this by induction from the fringe of the tree. For n an unexpanded node in T', the labels are identical in the two cases. If n is the node expanded to produce T' from T, the result follows immediately from the definition of a conservative labeling, since n will be labeled either with U (negative parity) or with $\emptyset$ (positive parity) in T.

For the inductive step, suppose that n is an internal node in the tree. If n is an action node, then the appropriate inequality holds for each of n's children by virtue of the inductive hypothesis, and therefore for n itself because of the definition (7). If n is a goal node, the result is a consequence of Lemma 2.4.

This completes the proof that $L_0^- \subseteq L_1^-$. To show that $L_0^- \subseteq L(g)$, we show the following:

1. The conservative labeling of T labels an action node $(a, +)$ with a plan that is a subset of the set of plans in which a has just been executed successfully.

2. The conservative labeling of T labels an action node $(a, -)$ with a superset of the set of plans in which a has just been executed successfully.

3. The conservative labeling of T labels a goal node $(g, +)$ with a subset of the set of plans in which g holds.

4. The conservative labeling of T labels a goal node $(g, -)$ with a superset of the set of plans in which g holds.

Once again, the proof proceeds by induction from the fringe of the tree; once again, that the above results hold for the fringe itself is obvious.

For an internal action node, the results hold because the set of plans in which a has just been executed successfully is exactly the intersection of the plans for which each of the preconditions of a holds, followed by the execution of the action a itself. For an internal goal node $(g, \pm)$, the set of plans for which g holds can be obtained by propagating the actions that successfully add or delete g. ■

Theorem 4.4 *Let g be a goal, and T_0 the planning tree consisting of the single goal node $(g, +)$. Now if p is a plan that achieves g, there is some expansion T of T_0 such that $p \in L^-(T)$. Conversely, if p fails to achieve g, there is an expansion such that $p \notin L^+(T)$.*

Proof. In fact, we show something stronger. Suppose that the length of p is k. Now we prove that if T is any tree with no unexpanded nodes shallower than depth $2k+2$, $p \in L^-(T)$ if and only if $p \in L^+(T)$. Since $L^-(T) \subseteq L(g) \subseteq L^+(T)$, the result will then follow.

The proof is by induction on k. If $k = 0$, so that $p = [\,]$, expanding the root node of T will produce an action node requiring *some* action (perhaps the initializing action i) to be taken to establish g. It follows that $[\,]$ will be in either both $L^-(T)$ and $L^+(T)$ or in neither.

For the inductive step, suppose that T has been fully expanded to depth $2k+2$. Now if n is a node in T of depth 2, we can apply the inductive hypothesis to conclude that L^- and L^+ agree on n for plans of length k or shorter. It follows that they will also agree on all (action) nodes of depth 1 and plans of length $k+1$ or less, since each of the depth 1 nodes requires the inclusion of a new final action. Since determining whether a plan p is in the set corresponding to the label for the root of the tree involves considering subsequences of p, the result for the root now follows from the result for the children at depth 1. ■

Proposition 5.4 *Suppose that T is a partially expanded planning tree for a goal g, and that n is a positive node in T. Now if there is any planning tree T' for g that matches T at n and for which $L^-(T') \neq \emptyset$, there is an expansion T_e of T with $L^-(T_e) \neq \emptyset$ such that T_e and T match at n.*

Proof. We can take T_e to be the union of T and T'. Since T_e is also an expansion of T', the result follows immediately from Theorem 4.3. ■

Proposition 7.1 *Any subgoal with no clobberers in its subtree is fortuitous.*

Proof. The situation is quite like that of Figure 4. Suppose that we label the repetition of the goal g with $\emptyset$ and that after doing so, conclude that $L(g) = P$. These are the plans that can be used for achieving g without recursion.

Now suppose we label the repeated appearance of g with L. Then the action node a_1 above g will be labeled with some subset of $L|a$. If the goal node above a_1 was previously labeled $Q = \pi(Q^+, \emptyset)$, it will now be labeled with a subset of

$$\begin{aligned} Q' &= \pi(Q^+ \cup L|a_1, \emptyset) = \pi(Q^+, \emptyset) \cup \pi(L|a_1, \emptyset) \\ &= Q \cup \pi(L|a_1, \emptyset) \end{aligned}$$

Continuing up the tree, if the path from the repeated occurrence of g back to the original one involves the actions $a_1, \ldots, a_n$, we see that the eventual label assigned to the root g will be a subset of

$$\pi(L|a_1|a_2|\ldots|a_n, \emptyset) \cup P$$

We therefore need to show that

$$\pi(P|a_1|a_2|\ldots|a_n, \emptyset) \cup P = P$$

or that

$$\pi(P|a_1|a_2|\ldots|a_n, \emptyset) \subseteq P$$

To see this, note that P itself is of the form $\pi(P^+, \emptyset)$ for some P^+, so that if we denote $\langle a_1, \ldots, a_n \rangle$ by a, we are trying to show

$$\pi(\pi(P^+, \emptyset)|a, \emptyset) \subseteq \pi(P^+, \emptyset) \tag{13}$$

But now let p be any element of the left hand side of (13). Since some initial subsequence of p is in $\pi(P^+, \emptyset)|a$, it follows that some initial subsequence of p is in $\pi(P^+, \emptyset)$, and therefore

that some initial subsequence of p is in P^+. Thus p itself is an element of $\pi(P^+, \emptyset)$, and (13) follows as desired. The proof is complete. ■

Acknowledgments

Effort sponsored by the Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory, Airforce Materiel Command, USAF, under agreements numbered F30602-95-1-0023 and F30602-97-1-0294. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon.

The views and conclusions contained herein are those of the author and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency (DARPA), the Air Force Research Laboratory, or the U.S. Government.

I would like to thank David Etherington, Oren Etzioni, David Joslin, Bart Massey, David McAllester, Dan Weld, and the members of CIRL for discussing these ideas with me.

References

[1] T. Bylander. The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, 69:165–204, 1994.

[2] D. Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32:333–377, 1987.

[3] J. M. Crawford and L. D. Auton. Experimental results on the crossover point in random 3sat. *Artificial Intelligence*, 81:31–57, 1996.

[4] K. S. Erol, D. S. Nau, and V. Subrahmanian. Complexity, decidability and undecidability results for domain-independent planning. *Artificial Intelligence*, 76:75–88, 1995.

[5] O. Etzioni. Acquiring search-control knowledge via static analysis. *Artificial Intelligence*, 62:255–302, 1993.

[6] O. Etzioni. A structural theory of explanation-based learning. *Artificial Intelligence*, 60:93–140, 1993.

[7] M. L. Ginsberg. Multivalued logics: A uniform approach to reasoning in artificial intelligence. *Computational Intelligence*, 4:265–316, 1988.

[8] M. L. Ginsberg. Bilattices and modal operators. *Journal of Logic and Computation*, 1:41–69, 1990.

[9] M. L. Ginsberg. Approximate planning. *Artificial Intelligence*, 76:89–123, 1995.

[10] M. L. Ginsberg. Modality and interrupts. *J. Automated Reasoning*, 14:43–91, 1995.

[11] A. K. Jönsson and M. L. Ginsberg. Efficient reasoning using procedures. In *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning*, Boston, MA, 1996.

[12] D. Joslin and M. Pollack. Passive and active decision postponement in plan generation. In *European Workshop on Planning*, 1995.

[13] R. E. Korf. Macro-operators: A weak method for learning. *Artificial Intelligence*, 26:35–77, 1985.

[14] B. Selman. Near-optimal plans, tractability, and reactivity. In *Proceedings of the Fourth International Conference on Principles of Knowledge Representation and Reasoning*, Bonn, Germany, 1994.

[15] D. E. Smith and M. A. Peot. Postponing threats in partial-order planning. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 500–506, 1993.

Appendix B

An Approach to Resource Constrained Project Scheduling*

James M. Crawford
CIRL
1269 University of Oregon
Eugene, OR 97403-1269
jc@cirl.uoregon.edu

Abstract

This paper gives an overview of a new approach to resource constrained project scheduling. The approach is based on the combination of a novel optimization technique with limited discrepancy search, and generates the best known solutions to benchmark problems of realistic size and character.

1 Introduction

Historically there has often been a mismatch between the types of scheduling problems that have been studied academically and the needs of the manufacturing community. The most obvious difference is that many real problems are much larger than common academic benchmarks. A second, and equally important, difference is that real problems generally involve constraints that have a more complex structure than can be expressed within a limited framework like job shop scheduling.

In this paper we overview ongoing work on resource constrained project scheduling (RCPS). RCPS is a generalization of job shop scheduling in which tasks can use multiple resources, and resources can have a capacity greater than one. RCPS is thus a good model for problems, like aircraft assembly, that cannot be expressed as job shop problems. In fact, if we take arguably the most widely used commercial scheduling program, Microsoft Project, RCPS seems to capture exactly the optimization problem that the "resource leveler" in Project solves.

It turns out that the algorithms that have been developed for job shop scheduling do not work particularly well for RCPS. In this paper we overview an approach to RCPS that is based on the combination of limited discrepancy search (LDS) with a novel optimization technique. The resulting system produces the best known schedules for problems of realistic size and character.

*This paper appeared in *Proc. 1996 Artificial Intelligence and Manufacturing Research Workshop.*

2 Resource Constrained Project Scheduling

A Resource Constrained Project Scheduling (RCPS) problem consists of a set of tasks, and a set of finite capacity resources. Each task puts some demand on the resources. For example, changing the oil might require one workman and one car lift. A partial ordering on the tasks is also given specifying that some tasks must preceded others (*e.g.*, you have to sand the board before you can paint it). Generally the goal is to minimize makespan without violating the precedence constraints or over-utilizing the resources.

RCPS is more general than job shop because resources can have capacity greater than one, and because tasks can use a collection of resources. This allows resources to be taken to be anything from scarce tools, to specialized workmen, to work zones (such as the cockpit of an airplane). RCPS problems arise in applications ranging from aircraft assembly to chemical refining.

Formally, Resource Constrained Project Scheduling is the following:

Given: A set of tasks T, a set of resources R, a capacity function $C : R \rightarrow \mathbb{N}$, a duration function $D : T \rightarrow N$, a utilization function $U : T \times R \rightarrow \mathbb{N}$, a partial order P on T, and a deadline d.

Find: An assignment of start times $S : T \rightarrow \mathbb{N}$, satisfying the following:

1. Precedence constraints: if t_1 precedes t_2 in the partial order P, then $S(t_1) + D(t_1) \leq S(t_2)$.
2. Resource constraints: For any time x, let $running(x) = \{t | S(t) \leq x < S(t) + D(t)\}$ Then for all times x, and all $r \in R$, $\sum_{t \in running(x)} U(t, r) \leq C(r)$.
3. Deadline: For all tasks $t : S(t) \geq 0$ and $S(t) + D(t) < d$.

3 A Benchmark Problem

Our experiments have been run on a series of problems made available on the WWW at:

`http://www.neosoft.com/~benchmrx`

by Barry Fox of McDonnell Douglas and Mark Ringer of Honeywell, serving as Benchmarks Secretary in the AAAI SIGMAN and in the AIAA AITC, respectively. These problems have 575 tasks and 17 resources. Some of the resources represent zone (geometric) constraints, and some represent labor constraints. Labor availability varies by shift. This is a synthetic problem that has been generated from experience with multiple large scale assembly problems. It is comparable to real problems is size and character, but simpler in the complexity of the constraints.

The results that have been posted to the WWW to date are shown in figure 1. Mark Ringer's results were found using a simple, first fit, interval based algorithm with no optimization. They were posted to encourage other contributions, rather than to generate the best solutions possible.

Who	Problem			Note
	2	3	4	
Mark Ringer	45	57	56	Honeywell
Nitin Agarwal	40	47	57	SAS
Colin Bell	39	-	-	Univ. of Iowa
Barry Fox	38	45	42	McDonnell Douglas
Crawford *et. al.*	38	43	41	CIRL
Crawford *et. al.*	38	39	38	(Lower Bound)

Figure 1: Results

4 Solution Methods

The two most important methods used in our scheduler are doubleback optimization and limited-discrepancy search. We discuss each in turn and then discuss how they work together in the scheduler.

4.1 Doubleback Optimization

The optimizer starts with any schedule satisfying the precedence constraints and generates a legal (and often a shorter) schedule. It works in two steps: a right shift and then a left shift (a very similar technique, *schedule packing* was independently invented previously by Barry Fox [1996]).

We first establish the right hand end point. Recall that the availability of some labor resources varies by shift. Because of this it turns out that it matters where in the daily cycle the endpoint is set.[1] The best heuristic seems to be to set it at the same point in the daily cycle as the end point of the current schedule. Another approach is to select the right hand end point randomly. This tends to shake things up a bit and makes iterating the optimizer more effective.

Once the right hand endpoint is selected we right shift the schedule. In the right shift we take the tasks in order of decreasing finish times (i.e., from the right hand end). We shift each task as far right as it will go. In doing this shift we consider only precedence constraints and resource constraints with previously shifted tasks (one way to think about this is to envision the new right hand endpoint as being at positive infinity: the tasks that have not yet been shifted do not interfere with the construction of the new schedule).

Once the right shift is completed, we left shift back to time zero, starting from the beginning of the right shifted schedule. As before, we left shift as far as possible subject to precedence constraints, and resource conflicts with previously left shifted tasks.

This sequence can be iterated. At some points this produces longer schedules (possibly then followed by shorter schedules after additional iterations). At present we have no theoretical method to predict the optimal number of iterations: we simply iterate ten times and keep the best schedule produced.

[1]This was first observed by Joe Pemberton.

To see the optimization works, consider the example shown in figure 2.

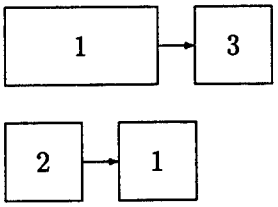


Figure 2: A simple scheduling problem.

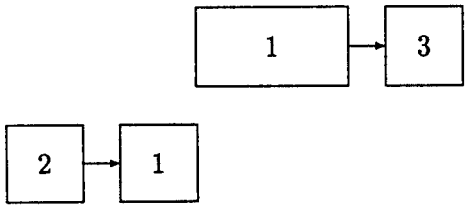


Figure 3: A bad schedule.

Here the boxes represent tasks and the arrows represent precedence relations. The numbers in the boxes are the resources the tasks need. For this example all resources have capacity one.

In order to break the resource conflict between the two tasks using resource one, we have to establish an ordering between the tasks. Assume that we do this non-optimally, generating the schedule shown in figure 3.[2]

Now consider what happens when we apply the optimizer. The right-shifted, and then left-shifted, schedules are shown in figure 4. The key thing to notice is that in the right shift the bottom task falls to the end of the schedule (because it has no successors), while the top task is forced to the beginning of the schedule. Thus the left shift schedules the top task first, generating the optimal schedule.

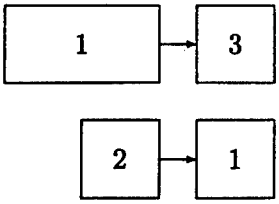
We can, of course, generate test cases in which the optimizer fails to find the shortest schedule, and we currently cannot offer any theoretical guarantees on the optimizer's performance. The strongest statement we can make is that on the benchmark examples, the optimizer is experimentally the single most effective scheduling technique we are aware of.

5 Why the Optimizer Works

Experimentally doubleback optimization is quite effective. Starting with the schedule that starts each task as early as possible subject to only the precedence constraints, the optimizer

[2]Such a mistake is unlikely in such a small problem, but our ability to avoid analogous mistakes in larger problems is, in a sense, the entire source of the intractability of scheduling.

Right shift:



Left shift:

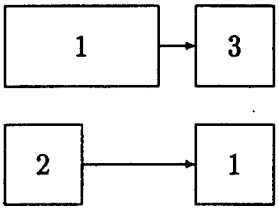


Figure 4: The effect of a right and then left shift.

is able to produce 43 day schedules for problem 4. The obvious question is why such a simple technique works so well.

In a sense the key decision to be made in scheduling is the ordering of tasks that compete for a resource. In fact the search portion of our approach (see below) essentially searches all possible ways to break resource contention by establishing an ordering between competing tasks.

The challenge of breaking resource contention is that we do not know which task will turn out to be the most important. In some cases it is obvious which tasks are most critical. For example, if a task must be followed by a long series of tasks, then clearly we want to give it a high priority – otherwise it will be postponed and the "tail" of tasks that follow it is likely to exceed the deadline. Unfortunately it is not generally this simple (or else scheduling would be tractable). In essence what goes wrong is that we do not know how hard it will be to schedule the sets of tasks that must follow the conflicting tasks. However, if we start from a "seed" schedule, then we can decide with reference to the seed, how hard the subsequent tasks will be, and use this information to make better decisions about task priorities.

It turns out that this is exactly what the optimizer does. The right shift pushes all tasks as late as possible. So, if a task is near the beginning of the right shifted schedule, it is there because it must be followed by a large number of tasks. So it should be given high priority. This is exactly what the left shift does: the left shifted schedule is formed by first schedule the tasks that are near the beginning of the right shifted schedule.

6 Limited Discrepancy Search

The results returned by the optimizer are sensitive to the "seed" schedule given to the optimizer. One can construct examples of "bad" schedules that the optimizer cannot correct. In a sense the optimizer is walking down to a kind of local minimum, and the quality of the final schedule depends on where the walk starts.

Our implementation uses LDS [Harvey & Ginsberg, 1995] to produce a series of seed schedules that are then passed to the optimizer. Here we give a brief overview of LDS. Details can be found in Harvey and Ginsberg [1995].

Imagine that we have a schedule that satisfies the precedence constraints, but not the resource constraints. The natural way to produce a legal schedule is to iteratively pick a resource conflict, delay one or more tasks long enough to break the conflict, and then propagate these delays through the precedence constraints. This is, in fact, how our current implementation works (starting from the left shifted schedule satisfying only the precedence constraints).

Each conflict that is broken creates a choice point, and breaking a series of conflicts produces a search tree. In the case of the benchmark problems this produces a search tree with a branching factor of about three, and a depth of about 1000.

The traditional approach to searching such a tree is to use a depth-first search. In a depth-first search, we make a series of decisions until we reach a leaf node in the tree (in this case a leaf node is a schedule satisfying both precedence and resource constraints). We then back up to the last choice point and take the other branch, and follow it to a leaf node. We then back up again, this time to the latest branch point that still has unexplored children. Repeating this we eventually search the entire tree, and are thus guaranteed to find the optimal schedule.

Unfortunately, if the search tree is 1000 nodes deep then a depth-first search will examine only a tiny fraction of the entire search tree (backing up perhaps 10 or 20 nodes). Further, it is reasonable to expect that the choices that will be reconsidered are exactly the choice for which the heuristic is most likely to have made the right decision. To see why this is so, notice that near the top of the search tree there are still many resource conflicts, so the heuristics are working from a "schedule" that is far from legal, so the heuristics are having to guess at how the resolution of these other conflicts will interact with the current conflict. Near the bottom of the tree, however, the schedule is in nearly its final form so the heuristics have good information on which to base their decisions.

As a result, traditional depth-first search is relatively little help on scheduling problems. This has lead many practitioners to either use no search (just following the heuristic and returning the first schedule produced) or to use a local search (which can reconsider any decision at any point).

In LDS we fix a bound on the number of times we will diverge from the heuristic. If that bound is zero then we just produce the single schedule given by always following the heuristics. If the bound is one then we produce a set of schedules generated by ignoring the heuristic exactly once.

The difference between LDS and depth-first search is illustrated in figures 5 and 6. In both search trees the branch preferred by the heuristic is always drawn on the left. In figure

5 the leaves are numbered according to the order in which depth-first search will visit them. Notice that if the heuristic makes a mistake high in the tree, for example, at the first choice point, then depth-first search will have to search half of the search tree before correcting the mistake. In the LDS search tree (figure 6), leaf nodes are labeled according to how many times the path from the root to the leaf diverges from the heuristic (*i.e.*, how many right turns are necessary to reach the leaf). LDS searches the leaves by first searching the leaf marked 0, then all the leaves marked 1, and so on.

If the heuristic is generally correct, but sometimes makes mistakes (as if generally the case with heuristics) then we can reasonably expect to find good quality schedules by searching the nodes for which the number of divergences is low. If the height of the tree is h, then the complexity of visiting each node with d divergences is h^d. In practice we usually set d to 1 or 2. This produces much better results than a depth-first search examining the same number of nodes. Further, unlike a local search, LDS is systematic: if we continue to raise d we are guaranteed to eventually find the optimal schedule.

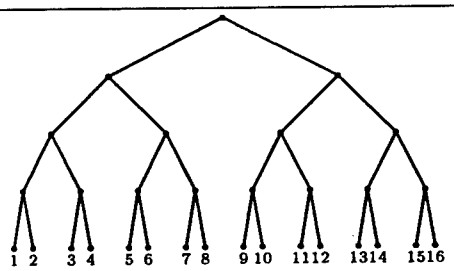


Figure 5: Backtracking search tree.

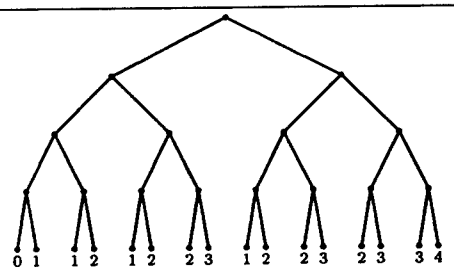


Figure 6: LDS search tree.

Finally we should node that LDS and the optimizer work well together. The optimizer is sensitive to the nature of the schedule it gets as input. Since LDS produces a series of reasonably good (but different) seed schedules, we can optimize each one, and in the end produce a schedule that is significantly shorter than we get by just optimizing the schedule given by following the heuristics exactly.

We can take this one step further and design the heuristic to avoid the kind of mistakes that the optimizer cannot fix.[3] This goes beyond the scope of the current paper, but it

[3]This idea came out of discussions with Matt Ginsberg.

turns out that we can identify certain kinds of task-ordering mistakes that a simple right-left shift is unable to untangle. We can then generate heuristics that will generally avoid these mistakes. This may cause us to find worse unoptimized schedules, but better schedules after optimization.

7 Conclusion

We have outlined an approach to RCPS problems that is based on using LDS to generate a series of "seed" schedules that are passed to an optimizer that can be seen as doing a kind of scheduling-specific local search. The results are currently the best known on problems of realistic size and character. Work continues on transitioning this technology to various application areas, and increasing the complexity of the constraints we can represent and effective optimize under.

Acknowledgment

This work has been supported by the Air Force Office of Scientific Research under grant number F49620-92-J-0384, by the Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory, Airforce Materiel Command, USAF, under agreement numbers F30602-93-C-00031 and F30602-95-1-0023, and by the National Science Foundation under grant number IRI-94 12205. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon.

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency (DARPA), the Air Force Research Laboratory, or the U.S. Government.

The work was done at the Computational Intelligence Research Laboratory, and owes much to discussions with all the members of the lab, particularly Matt Ginsberg, Joe Pemberton, and Ari Jonsson. Finally, we should add that this work could not have been done without the effort Barry Fox and Mark Ringer have put in to make realistic benchmark problems available on the WWW.

References

[Fox, 1996] Fox, B. 1996. An algorithm for scheduling improvement by scheduling shifting. Technical Report 96.5.1, McDonnell Douglas Aerospace - Houston. McDonnell Douglas has applied for a patent on this work.

[Harvey & Ginsberg, 1995] Harvey, W. D., and Ginsberg, M. L. 1995. Limited discrepancy search. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, volume 1, 607–613.

Appendix C

Tuning Local Search for Satisfiability Testing*

Andrew J. Parkes
CIS Dept. and CIRL
1269 University of Oregon
Eugene, OR 97403-1269
U.S.A.
parkes@cirl.uoregon.edu

Joachim P. Walser
Programming Systems Lab
Universität des Saarlandes
Postfach 151150, 66041 Saarbrücken
Germany
walser@cs.uni-sb.de

February 4, 1999

Abstract

Local search algorithms, particularly GSAT and WSAT, have attracted considerable recent attention, primarily because they are the best known approaches to several hard classes of satisfiability problems. However, replicating reported results has been difficult because the setting of certain key parameters is something of an art, and because details of the algorithms, not discussed in the published papers, can have a large impact on performance. In this paper we present an efficient probabilistic method for finding the optimal setting for a critical local search parameter, Maxflips, and discuss important details of two differing versions of WSAT. We then apply the optimization method to study performance of WSAT on satisfiable instances of Random 3SAT at the crossover point and present extensive experimental results over a wide range of problem sizes. We find that the results are well described by having the optimal value of Maxflips scale as a simple power of the number of variables, n, and the average run time scale sub-exponentially (basically as $n^{\log(n)}$) over the range $n = 25, \ldots, 400$.

1 INTRODUCTION

In recent years, a variety of local search routines have been proposed for (Boolean) satisfiability testing. It has been shown [Selman, Levesque, & Mitchell, 1992; Gu, 1992; Selman, Kautz, & Cohen, 1994] that local search can solve a variety of realistic and randomly generated satisfiability problems much larger than conventional procedures such as Davis-Putnam.

The characteristic feature of local search is that it starts on a total variable assignment and works by repeatedly changing variables that appear in violated constraints [Minton *et al.*, 1990]. The changes are typically made according to some hill-climbing heuristic strategy with the aim of maximizing the number of satisfied constraints. However, as is usual with hill-climbing, we are liable to get stuck on local maxima. There are two standard ways to overcome this problem: "noise" can be introduced [Selman, Kautz, & Cohen, 1994]; or,

*This paper appeared in *Proc. AAAI-96.*

with a frequency controlled by a *cutoff* parameter Maxflips we can just give up on local moves and restart with some new assignment. Typically both of these techniques are used together but their use raises several interesting issues:

1. What value should we assign to Maxflips? There are no known fundamental rules for how to set it, yet it can have a significant impact on performance and deserves optimization. Also, empirical comparison of different procedures should be done fairly, which involves first optimizing parameters.

2. If we cannot make an optimal choice then how much will performance suffer?

3. How does the performance scale with the problem size? This is especially important when comparing local search to other algorithm classes.

4. Local search routines can fail to find a solution even when the problem instance is actually satisfiable. We might like an idea of how often this happens, i.e. the false failure rate under the relevant time and problem size restrictions.

At present, resolving these issues requires extensive empirical analysis because the random noise implies that different runs can require very different runtimes even on the same problem instance. Meaningful results will require an average over many runs. In this paper we give a probabilistic method to reduce the computational cost involved in Maxflips optimization and also present scaling results obtained with its help.

The paper is organized as follows: Firstly, we discuss a generic form of a local search procedure, and present details of two specific cases of the WSAT algorithm [Selman, Kautz, & Cohen, 1994]. Then we describe the optimization method, which we call "*retrospective parameter variation*" (RPV), and show how it allows data collected at one value of Maxflips to be reused to produce runtime results for a range of values. We note that the same concept of RPV can also be used to study the effects of introducing varying amounts of parallelization into local search by simulating multiple threads [Walser, 1995].

Finally, we present the results of experiments to study the performance of the two versions of WSAT on Random 3SAT at the crossover point [Cheeseman, Kanefsky, & Taylor, 1991; Mitchell, Selman, & Levesque, 1992; Crawford & Auton, 1996]. By making extensive use of RPV, and fast multi-processor machines, we are able to give results up to 400 variables. We find that the optimal Maxflips setting scales as a simple monomial, and the mean runtime scales subexponentially, but faster than a simple power-law.

2 LOCAL SEARCH IN SAT

Figure 1 gives the outline of a typical local search routine [Selman, Levesque, & Mitchell, 1992] to find a satisfying assignment for a set of clauses α[1].

Here, local moves are "flips" of variables that are chosen by *select-variable*, usually according to a randomized greedy strategy. We refer to the sequence of flips between restarts (new total truth assignments) as a "try", and a sequence of tries finishing with

[1] A clause is a disjunction of literals. A literal is a propositional variable or its negation.

```
proc Local-Search-SAT
    Input clauses α, Maxflips, and Maxtries
    for i := 1 to Maxtries do
        A := new total truth assignment
        for j := 1 to Maxflips do
            if A satisfies α then return A
            P := select-variable(α, A)
            A := A with P flipped
        end
    end
    return "No satisfying assignment found"
end
```

Figure 1: A generic local search procedure for SAT.

a successful try as a "run". The parameter Maxtries can be used to ensure termination (though in our experiments we always set it to infinity). We also assume that the *new* assignments are all chosen randomly, though other methods have been considered [Gent & Walsh, 1993].

2.1 Two WSAT Procedures

In our experiments, we used two variants of the WSAT – "walk" satisfiability class of local search procedures. This class was introduced by Selman et. al. [Selman, Kautz, & Cohen, 1994] as "WSAT makes flips by first randomly picking a clause that is not satisfied by the current assignment, and then picking (either at random or according to a greedy heuristic) a variable within that clause to flip." Thus WSAT is a restricted version of Figure 1 but there remains substantial freedom in the choice of heuristic. In this paper we focus on the two selection strategies, as given in Figure 2. Here, f_P denotes the number of clauses that are *fixed* (become satisfied) if variable P is flipped. Similarly, b_P is the number that *break* (become unsatisfied) if P is flipped. Hence, $f_P - b_P$ is simply the net increase in the number of satisfied clauses.

The first strategy[2] WSAT/G is simple hillclimbing on the net number of satisfied clauses, but perturbed by noise because with probability p, a variable is picked randomly from the clause. The second procedure WSAT/SKC is that of a version of WSAT by Cohen, Kautz, and Selman.[3] We give the details here because they were not present in the published paper [Selman, Kautz, & Cohen, 1994], but are none-the-less rather interesting. In particular, WSAT/SKC uses a less obvious, though very effective, selection strategy. Firstly, hill-climbing is done solely on the number of clauses that *break* if a variable is flipped, and the number of clauses that get fixed is ignored. Secondly, a random move is never made if it is possible to do a move in which no previously satisfied clauses become broken. In all it exhibits a sort of "minimal greediness", in that it definitely fixes the one randomly selected

[2]Andrew Baker, personal communication.

[3]Publically available, ftp://ftp.research.att.com/dist/ai

WSAT/G

proc *select-variable*(α, A)
  $C :=$ a random unsatisfied clause
  with probability p :
    $S :=$ random variable in C
  probability $1-p$:
    $S :=$ variable in C with maximal $f_P - b_P$
  end
  return S
end

WSAT/SKC

proc *select-variable*(α, A)
  $C :=$ a random unsatisfied clause
  $u := \min_{S \in C} b_S$
  if $u = 0$ **then**
    $S :=$ a variable $P \in C$ with $b_P = 0$
  else
    with probability p :
      $S :=$ random variable in C
    probability $1-p$:
      $S :=$ variable $P \in C$ with minimal b_P
    end
  return S
end

Figure 2: Two WSAT variable selection strategies. Ties for the best variable are broken at random.

clause but otherwise merely tries to minimize the damage to the already satisfied clauses. In contrast, WSAT/G is greedier and will blithely cause lots of damage if it can get paid back by other clauses.

3 RETROSPECTIVE VARIATION OF MAXFLIPS

We now describe a simple probabilistic method for efficiently determining the Maxflips dependence of the mean runtime of a randomized local search procedure such as WSAT. We use the term "retrospective" because the parameter is varied after the actual experiment is over.

As discussed earlier, a side-effect of the randomness introduced into local search procedures is that the runtime now varies between runs. It is often the case that this "*inter-run*" variation is large and to give meaningful runtime results we need an average over many runs. Furthermore, we will need to determine the mean runtime over a range of values of Maxflips. The naive way to proceed is to do totally independent sets of runs at many

different Maxflips values. However, this is rather wasteful of data because the successful try on a run often uses many fewer flips than the current Maxflips, and so we should be able to re-use it (together with the number of failed tries) to produce results for smaller values of Maxflips.

Suppose we take a *single* problem instance ν and make many runs of a local search procedure with Maxflips=m_D, resulting in a sample with a total of N tries. The goal is to make predictions for Maxflips = $m < m_D$. Label the *successful* tries by i, and let x_i be the number of flips it took i to succeed. Write the bag of all successful tries as $S_0 = \{x_1, \ldots, x_l\}$ and define a reduced bag S_0^m by removing tries that took longer than m

$$S_0^m := \{x_i \in S_0 \mid x_i \leq m\}. \tag{1}$$

We are asssuming there is randomness in each try and no learning between tries so we can consider the tries to be independent. From this it follows that the new bag provides us with information on the distribution of flips for successful tries with Maxflips=m. Also, an estimate for the probability that a try will succeed within m flips is simply

$$p_m \approx \frac{|S_0^m|}{N} \tag{2}$$

Together S_0^m and p_m allow us to make predictions for the behaviour at Maxflips=m.

In this paper we are concerned with the expected (mean) number of flips $E_{\nu,m}$ for the instance ν under consideration. Let $\overline{S_0^m}$ be the mean of the elements in the bag S_0^m. With probability p_m, the solution will be found on the first try, in which case we expect $\overline{S_0^m}$ flips. With probability $(1 - p_m)\, p_m$, the first try will fail, but the second will succeed, in which case we expect $m + \overline{S_0^m}$ flips, and so on. Hence,

$$E_{\nu,m} = \sum_{k=0}^{\infty} (1 - p_m)^k \, p_m \, (k\, m + \overline{S_0^m}) \tag{3}$$

which simplifies to give the main result of this section

$$E_{\nu,m} = (1/p_m - 1)\, m + \overline{S_0^m} \tag{4}$$

with p_m and $\overline{S_0^m}$ estimated from the reduced bag as defined above. This is as to be expected since $1/p_m$ is the expected number of tries. It is clearly easy to implement a system to take single data-set obtained at Maxflips=m_D, and estimate the expected number of flips for many different smaller values of m.

Note that it might seem that a more direct and obvious method would be to take the bag of all *runs* rather than *tries*, and then simply discard runs in which the final try took longer than m. However, such a method would discard the information that the associated tries all took longer than m. In contrast, our method captures this information: the entire population of tries is used.

Instance Collections To deal with a collection, C, of instances we apply RPV to each instance individually and then proceed exactly as if this retrospectively simulated data had been obtained directly. For example, the expected mean number of flips for C is

$$E_m = \frac{1}{|C|} \sum_{\nu \in C} E_{\nu,m}. \tag{5}$$

Note that this RPV approach is not restricted to means, but also allows the investigation of other statistical measures such as standard deviation or percentiles.

Practical Application of RPV The primary limit on the use of RPV arises from the need to ensure that the bag of successful tries does not get too small and invalidate the estimate for p_m. Since the bag size decreases as we decrease m it follows that there will be an effective lower bound on the range over which we can safely apply RPV from a given m_D. This can be offset by collecting more runs per instance. However, there is a tradeoff to be made: If trying to make predictions at too small a value of m it becomes more efficient to give up on trying to use the data from Maxflips=m_D and instead make a new data collection at smaller Maxflips. This problem with the bag size is exacerbated by the fact that different instances can have very different behaviours and hence different ranges over which RPV is valid. It would certainly be possible to do some analysis of the errors arising from the RPV. The data collection system could even do such an analysis to monitor current progress and then concentrate new runs on the instances and values of Maxflips for which the results are most needed. In practice, we followed a simpler route: we made a fixed number of runs per instance and then accepted the RPV results only down to values of m for some fixed large fraction of instances still had a large enough bagsize.

Hence, RPV does not always remove the need to consider data collection at various values of Maxflips, however, it does allow us to collect data at more widely separated Maxflips values and then interpolate between the resulting "direct" data points: This saves a time-consuming fine-grained data-collection, or binary search through Maxflips values.

4 EXPERIMENTAL RESULTS

To evaluate performance of satisfiability procedures, a class of randomized benchmark problems, Random 3SAT, has been studied extensively [Mitchell, Selman, & Levesque, 1992; Mitchell, 1993; Crawford & Auton, 1996]. Random 3SAT provides a ready source of hard scalable problems. Problems in random k-SAT with n variables and l clauses are generated as follows: a random subset of size k of the n variables is selected for each clause, and each variable negated with probability 1/2. If instances are taken from near the crossover point (where 50% of the randomly generated problems are satisfiable) then the fastest systematic algorithms, such as TABLEAU [Crawford & Auton, 1996], show a well-behaved increase in hardness: time required scales as a simple exponential in n.

In the following we present results for the performance of both variants of WSAT on satisfiable Random 3SAT problems at the crossover point. We put particular emphasis on finding the Maxflips value m^* at which the mean runtime averaged over all instances is a minimum. Note that the clause/variable ratio is not quite constant at the crossover point but tends to be slightly higher for small n. Hence, to avoid "falling off the hardness peak", we used the experimental results [Crawford & Auton, 1996] for the number of clauses at crossover, rather than using a constant value such as $4.3n$. To guarantee a fair sample of satisfiable instances we used TABLEAU to filter out the unsatisfiable instances. At n=400 this took about 2-4 hours per instance, and so even this part was computationally non-trivial, and in fact turned out to be the limiting factor for the maximum problem size.

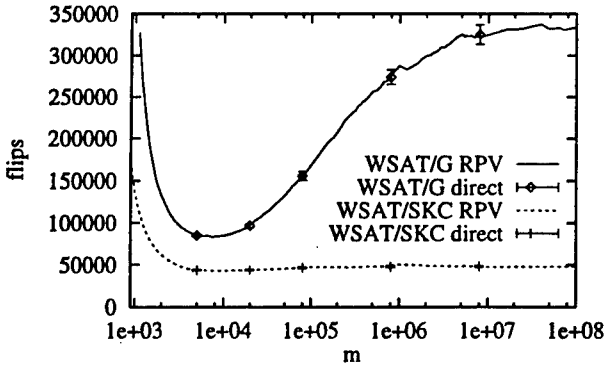


Figure 3: Variation of E_m against Maxflips=m. Based on 1k instances of (200,854) with 200 runs per instance.

For WSAT/SKC, the setting of the noise parameter p has been reported to be optimal between 0.5 and 0.6 [Selman, Kautz, & Cohen, 1994]. We found evidence that such values are also close to optimal for WSAT/G, hence we have produced all results here with $p = 0.5$ for both WSAT variants, but will discuss this further in the next section.

We present the experiments in three parts. Firstly, we compare the two variants of WSAT using a small number of problem instances but over a wide range of Maxflips values to show the usage of RPV to determine their Maxflips dependencies. We then concentrate on the more efficient WSAT/SKC, using a medium number of instances, and investigate the scaling properties over the range $n = 25, \ldots, 400$ variables. This part represents the bulk of the data collection, and heavily relied on RPV. Finally, we look at a large data sample at $n = 200$ to check for outliers.

4.1 Overall Maxflips Dependence

Our aim here is to show the usage of RPV and also give a broad picture of how the mean runtime E_m varies with m for the two different WSAT variants. We took a fixed, but randomly selected, sample of 10^3 instances at $(n, l) = (200, 854)$, for which we made 200 runs on each instance at m=5k, 20k, 80k, 800k, 8000k, ∞ using both variants of WSAT. At each m we directly calculated E_m. We then applied RPV to the samples to extend the results down towards the next data-point.

The results are plotted in Figure 3. Here the error bars are the 95% confidence intervals for the particular set of instances, i.e. they reflect only the uncertainty from the limited number of runs, and *not* from the limited number of instances. Note that the RPV lines from one data point do indeed match the lower, directly obtained, points. This shows that the RPV is not introducing significant errors when used to extrapolate over these ranges.

Clearly, at $p = 0.5$, the two WSAT algorithms exhibit very different dependencies on Maxflips: selecting too large a value slows WSAT/G down by a factor of about 4, in contrast to a slowdown of about 20% for WSAT/SKC. At Maxflips=∞ the slowest try for WSAT/G took 90 minutes against 5 minutes for the worst effort from WSAT/SKC. As has been

observed before [Gent & Walsh, 1995], "random walk" can significantly reduce the Maxflips sensitivity of a local search procedure: Restarts and noise fulfill a similar purpose by allowing for downhill moves and driving the algorithm around in the search space. Experimentally we found that while the peak performance E_{m^*} varies only very little with small variation of p (± 0.05), the Maxflips sensitivity can vary quite remarkably. This topic warrants further study and again RPV is useful since it effectively reduces the two-dimensional parameter space (p, m) to just p.

While the average difference for E_{m^*} between the two WSATs on Random 3SAT is typically about a factor of two, we found certain instances of circuit synthesis problems [Selman, Kautz, & Cohen, 1994] where WSAT/SKC is between 15 and 50 times faster. Having identified the better WSAT, we will next be concerned with its optimized scaling.

4.2 Extensive Experiments on WSAT/SKC

We would now like to obtain accurate results for the performance of WSAT/ SKC on the Random 3SAT domain. So far, we have only considered confidence intervals resulting from the inter-run variation. Unfortunately, for the previous sample of 1000 instances, the error in E_m arising from the inter-instance variation is much larger (about 25%). This means that to obtain reasonable total errors we needed to look at larger samples of instances.

We were mostly interested in the behaviour at and near m^*. Preliminary experiments indicated that making the main data collection at just $m = 0.5n^2$ would allow us to safely use RPV to study the minimum. Hence, using WSAT/SKC at $p = 0.5$, we did 200 runs per instance at $m = 0.5n^2$, and then used RPV to find m^*. The results are summarized in Table 1, and we now explain the other entries in this table. We found it useful to characterize the E_m curves by values of Maxflips which we denote by m_{k-}, and define as the largest value of m such that $m < m^*$ and E_m is $k\%$ larger than E_{m^*}. Similarly we define m_{k+} as the smallest value of $m > m^*$ with the same property. We can easily read these off from the curve produced by RPV. The RPV actually produces the set E_{ν,m^*} and so we also sorted these and calculated various percentiles of the distribution (the 99th percentile means that we expect that 99% of the instances will, on average, be solved in this number of flips). Finally, the error on E_{m^*} is the 95% confidence level as obtained from the standard deviation of the $E_{\nu,m}$ sample (inter-instance). The error from the limited number of runs was negligible in comparison. In the next section we interpret this data with respect to how m^* and E_{m^*} vary with n. We did not convert flips to times because the actual flips rate varies remarkably little (from about 70k-flips/sec down to about 60k-flips/sec).

4.3 Scaling of Optimal Maxflips

In Figure 4 we can see how m^* and m_{5-} vary with n. In order to interpret this data we fitted the function an^b against m_{5-} because the E_m curves are rather flat and so m^* is relatively ill-defined. However, they seem to have the same scaling and also $m^* > m_{5-}$ by definition. We obtained a best fit[4] with the values $a = 0.02$ and $b = 2.39$ (the resulting line

[4]Using the Marquardt-Levenberg algorithm as implemented in Gnufit by C. Grammes at the Universität des Saarlandes, Germany, 1993.

Vars	Cls	m^*	m_{5-}	E_{m^*}	95%-cnf.	Median	99-perc.
25	113	70	37	116	2	94	398
50	218	375	190	591	12	414	2,876
100	430	2,100	1,025	3,817	111	2,123	27,367
150	641	6,100	2,925	13,403	486	5,891	120,597
200	854	11,900	5,600	36,973	2,139	12,915	406,966
250	1066	15,875	8,800	92,915	8,128	25,575	1,050,104
300	1279	23,200	13,100	171,991	15,455	43,314	2,121,809
350	1491	32,000	19,300	334,361	69,850	65,574	4,258,904
400	1704	43,500	27,200	528,545	114,899	96,048	11,439,288

Table 1: Experimental results for WSAT/SKC with $p = 0.5$ on Random 3SAT at crossover. The results are based on 10k instances (25–250 variables), 6k instances (300 vars), 3k instances (350 vars) and 1k instances (400 vars).

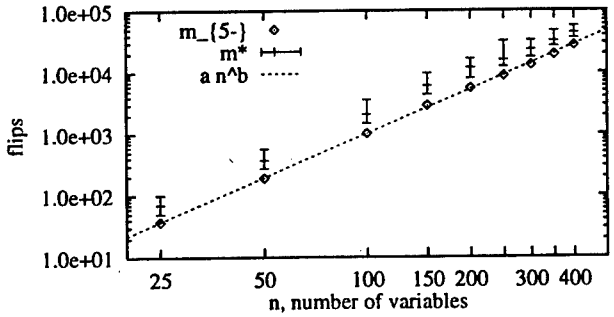


Figure 4: The variation of m_{5-} and m^* with n. We use m_{1+} and m_{1-} as the upper and lower error bounds on m^*.

is also plotted in Figure 4). Similar results for WSAT/G also indicate a very similar scaling of $n^{2.36}$. For comparison HSAT, a non-randomized GSAT variant that incorporates a history mechanism, has been observed to have a m^* scaling of $n^{1.65}$ [Gent & Walsh, 1995].

4.4 Scaling of Performance

In Figure 5 we plot the variation of E_{m^*} with n. We can see that the scaling is not as fast as a simple exponential in n, however the upward curve of the corresponding log-log plot (not presented) indicates that it is also worse than a simple monomial. Unfortunately, we know of no theoretical scaling result that could be reasonably compared with this data. For example, results are known [Koutsoupias & Papadimitriou, 1992] when the number of clauses is $\Omega(n^2)$, but they do not apply because the crossover is at $O(n)$. Hence we approached the scaling from a purely empirical perspective, by trying to fit functions to the data that can reveal certain characteristics of the scaling. We also find it very interesting that m^* so often seems to fit a simple power law, but are not aware of any explanation for this. However, the fits do provide an idea of the scaling that might have practical use, or

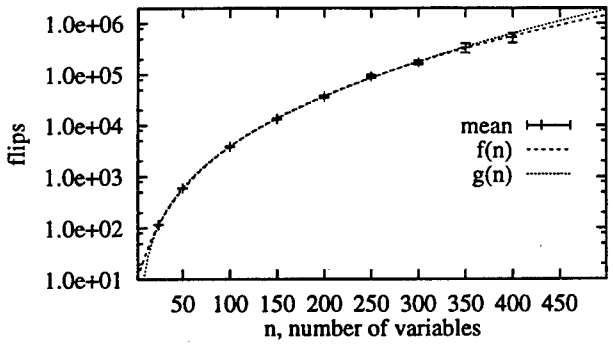


Figure 5: The scaling behaviour of WSAT/SKC at crossover. The data points are E_{m^*} together with its 95% confidence limits. The lines are best fits of the functions given in the text.

maybe even lead to some theoretical arguments.

We could find no good 2-parameter fit to the E_{m^*} curve, however the following functions give good fits, and provide the lines on Figure 5. (For $f(n)$, we found the values $f_1 = 12.5 \pm 2.02$, $f_2 = -0.6 \pm 0.07$, and $f_3 = 0.4 \pm 0.01$.)

$$\begin{aligned} f(n) &= f_1 n^{f_2 + f_3 \lg(n)} \\ g(n) &= g_1 \exp(n^{g_2}(1 + g_3/n)) \end{aligned}$$

The fact that such different functions give good fits to the data illustrates the obviously very limited discriminating power of such attempts at empirical fits. We certainly do not claim that these are asymptotic complexities! However, we include them as indicative. The fact that $f_3 > 0$ indicates a scaling which is similar to a simple power law except that the exponent is slowly growing. Alternatively, the fact that we found $g_2 \approx 0.4$, can be regarded as a further indication that scaling is slower than a simple exponential (which would have $g_2 = 1$).

4.5 Testing for Outliers

One immediate concern is whether the average runtimes quoted above are truly meaningful for the problem class Random 3SAT. It could easily be that the effect of outliers, instances that WSAT takes much longer to solve, eventually dominates. That is, as the sample size increases then we could get sufficiently hard instances and with sufficient frequency such that the mean would drift upwards [Mitchell, 1993].

To check for this effect we decided to concentrate on $(n, l) = (200, 854)$ and took 10^5 instances. Since we only wanted to check for outliers, we did not need high accuracy estimates and it was sufficient to do just 20 runs/instance of WSAT/SKC at Maxflips=80k, not using RPV. We directly calculated the mean for each seed: in Figure 6 we plot the distribution of the $\lg(E_{\nu,m})$.

If the tail of the distribution were too long, or if there were signs of a bimodal distribution with a small secondary peak but at a very large E_{m^*} then we would be concerned about

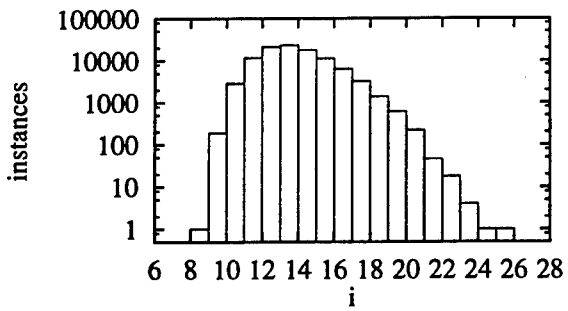


Figure 6: From a total of 10^5 instances we give the number of instances for which the expected number of flips $E_{\nu,m}$ is in the range $2^i \leq E_{\nu,m} < 2^{i+1}$. This is for WSAT/SKC at $m = 80$k, based on 20 runs per instance.

the validity of the means quoted above. However we see no signs of any such effects. On the contrary the distribution seems to be quite smooth, and its mean is consistent with the previous data. However, we do note that the distribution tail is sufficiently large that a significant fraction of total runtime is spent on relatively few instances. This means that sample sizes are effectively reduced, and is the reason for our use of 10k samples for the scaling results where computationally feasible.

5 RELATED WORK

Optimal parameter setting of Maxflips and scaling results have been examined before by Gent and Walsh [Gent & Walsh, 1993; Gent & Walsh, 1995]. However, for each sample point (each n) Maxflips had to be optimized experimentally. Thus, the computational cost of a systematic Maxflips optimization for randomized algorithms was too high for problems larger than 100 variables. This experimental range was not sufficient to rule out a polynomial runtime dependence of about order 3 (in the number of variables) for GSAT [Gent & Walsh, 1993].

A sophisticated approach to the study of incomplete randomized algorithms is the framework of Las Vegas algorithms which has been used by Luby, Sinclair, and Zuckermann [Luby, Sinclair, & Zuckerman, 1993] to examine optimal speedup. They have shown that for a *single* instance there is no advantage to having Maxflips vary between tries and present optimal cutoff times based on the distribution of the runtimes. These results, however, are not directly applicable to average runtimes for a collection of instances.

Local Search procedures also have close relations to simulated annealing [Selman & Kautz, 1993]. Indeed, combinations of simulated annealing with GSAT have been tried for hard SAT problems [Spears, 1995]. We can even look upon the restart as being a short period of very high temperature that will drive the variable assignment to a random value. In this case we find it interesting that work in simulated annealing also has cases in which periodic reheating is useful [Boese & Kahng, 1994]. We intend to explore these connections further.

6 CONCLUSIONS

We have tried to address the four issues in the introduction empirically. In order to allow for optimizing Maxflips, we presented retrospective parameter variation (RPV), a simple resampling method that significantly reduces the amount of experimentation needed to optimize certain local search parameters.

We then studied two different variants of WSAT, which we label as WSAT/G and WSAT/SKC (the latter due to Selman et al.). The application of RPV revealed better performance of WSAT/SKC. An open question is whether the relative insensitivity of WSAT/SKC to restarts carries over to realistic problem domains. We should note that for general SAT problems the Maxflips-scaling is of course not a simple function of the number of variables and p only, but is also affected by other properties such as the problem constrainedness.

To study the scaling behaviour of local search, we experimented with WSAT/SKC on hard Random 3SAT problems over a wide range of problem sizes, applying RPV to optimize performance. Our experimental results strongly suggest subexponential scaling on Random 3SAT, and we can thus support previous claims [Selman, Levesque, & Mitchell, 1992; Gent & Walsh, 1993] that local search scales significantly better than Davis-Putnam related procedures.

Unfortunately RPV cannot be used to directly determine the impact of the noise parameter p, however it is still useful since instead of varying two parameters, only p has to be varied experimentally, while different Maxflips values can be simulated.

We plan to extend this work in two further directions. Firstly, we intend to move closer to real problems. For example, investigations of binary encodings of scheduling problems reveal a strong correlation between the number of bottlenecks in the problems and optimal Maxflips for WSAT/G. Secondly, we would like to understand the Maxflips-dependence in terms of behaviours of individual instances.

ACKNOWLEDGEMENTS

Effort sponsored in part by the Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory, Airforce Materiel Command, USAF, under agreements numbered F30602-93-C-0031 and F30602-95-1-0023, and by a doctoral fellowship of the DFG to the second author (Graduiertenkolleg Kognitionswissenschaft). The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon.

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency (DARPA), the Air Force Research Laboratory, or the U.S. Government.

We would like to thank the members of CIRL, particularly Jimi Crawford, for many helpful discussions, and Andrew Baker for the use of his WSAT. We thank David McAllester and the anonymous reviewers for very helpful comments on this paper. The experiments reported here were made possible by the use of three SGI (Silicon Graphic Inc.) Power

Challenge systems purchased by the University of Oregon Computer Science Department through NSF grant STI-9413532.

References

[Boese & Kahng, 1994] Boese, K., and Kahng, A. 1994. Best-so-far vs. where-you-are: implications for optimal finite-time annealing. *Systems and Control Letters* 22(1):71–8.

[Cheeseman, Kanefsky, & Taylor, 1991] Cheeseman, P.; Kanefsky, B.; and Taylor, W. 1991. Where the really hard problems are. In *Proceedings IJCAI-91.*

[Crawford & Auton, 1996] Crawford, J., and Auton, L. 1996. Experimental results on the crossover point in Random 3SAT. *Artificial Intelligence.* To appear.

[Gent & Walsh, 1993] Gent, I., and Walsh, T. 1993. Towards an understanding of hill-climbing procedures for SAT. In *Proceedings AAAI-93*, 28–33.

[Gent & Walsh, 1995] Gent, I., and Walsh, T. 1995. Unsatisfied variables in local search. In *Hybrid Problems, Hybrid Solutions (Proceedings of AISB-95).* IOS Press.

[Gu, 1992] Gu, J. 1992. Efficient local search for very large-scale satisfiability problems. *SIGART Bulletin* 3(1):8–12.

[Koutsoupias & Papadimitriou, 1992] Koutsoupias, E., and Papadimitriou, C. 1992. On the greedy algorithm for satisfiability. *Information Processing Letters* 43:53–55.

[Luby, Sinclair, & Zuckerman, 1993] Luby, M.; Sinclair, A.; and Zuckerman, D. 1993. Optimal speedup of Las Vegas algorithms. Technical Report TR-93-010, International Computer Science Institute, Berkeley, CA.

[Minton *et al.*, 1990] Minton, S.; Johnston, M. D.; Philips, A. B.; and Laird, P. 1990. Solving large-scale constraint satisfcation and scheduling problems using a heuristic repair method. *Artificial Intelligence* 58:161–205.

[Mitchell, Selman, & Levesque, 1992] Mitchell, D.; Selman, B.; and Levesque, H. 1992. Hard and easy distributions of SAT problems. In *Proceedings AAAI-92*, 459–465.

[Mitchell, 1993] Mitchell, D. 1993. An empirical study of random SAT. Master's thesis, Simon Fraser University.

[Selman & Kautz, 1993] Selman, B., and Kautz, H. 1993. An empirical study of greedy local search for satisfiability testing. In *Proceedings of IJCAI-93.*

[Selman, Kautz, & Cohen, 1994] Selman, B.; Kautz, H.; and Cohen, B. 1994. Noise strategies for improving local search. In *Proceedings AAAI-94*, 337–343.

[Selman, Levesque, & Mitchell, 1992] Selman, B.; Levesque, H.; and Mitchell, D. 1992. A new method for solving hard satisfiability problems. In *Proceedings AAAI-92*, 440–446.

[Spears, 1995] Spears, W. 1995. Simulated annealing for hard satisfiability problems. *DIMACS Series on Discrete Mathematics and Theoretical Computer Science.* To appear.

[Walser, 1995] Walser, J. 1995. Retrospective analysis: Refinements of local search for satisfiability testing. Master's thesis, University of Oregon.

Appendix D

The CSPC Compiler User's Manual*†

Ari K. Jónsson

1 Introduction

1.1 Purpose

Constraint satisfaction problems frequently appear as subproblems in artificial intelligence and various other areas of computer science. In recent years, significant advances have been made in the area of constraint satisfaction, ranging from faster solvers to clearer theoretical understanding of the behavior of such problems.

Much of this work has been concentrated on a subset of constraint satisfaction problems, so called satisfiability (SAT) problems. A satisfiability problem is defined by a finite set of boolean variables and constraints in the form of clauses.

The major obstacle to applying the recent advance to real world problems, is the amount of work it takes to translate a problem into a constraint satisfaction problem. Two approaches have been used in the past. One is to write the constraints into the solver, thus creating a solver that handles a specific subset of CSPs. The other is to write a translator that maps a specific set of CSPs into SAT problems. The main problem with both approaches is that a new translator or solver must be written each time a new set of problems is attacked. Furthermore, a fair amount of cleverness is needed to make the mapping into a SAT problem efficient.

The `cspc` compiler solves these problems by compiling problem descriptions, given in a high level first order language, into lower level input formats for various problem solvers. This makes it unnecessary to write new translators or solvers when a new class of problems is encountered. Furthermore, the mapping into SAT problems can automatically be made more efficient, by letting the compiler eliminate variables and constraints that can be proven to be redundant.

*Effort sponsored by the Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory, Airforce Materiel Command, USAF, under agreement numbered F30602-95-1-0023. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon.

†The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency (DARPA), the Air Force Research Laboratory, or the U.S. Government.

1.2 Description

The compiler reads a problem description, written in the first order language described below, and compiles it into a satisfiability problem, or a general constraint satisfaction problem. The SAT problems are represented in a standardized fashion, making it possible to use any of the modern SAT engines available, tableau, wsat, etc. The CSP problems are represented in a new language that represents them efficiently, without making it difficult for a search engine to handle the variables, domains and constraints.

There are other two important advantages to using the `cspc` compiler. One is that it allows the user to have the symbolic names of variables recorded, so that a solution can be represented symbolically to the user. The other is that it can handle the specification of extension procedures, making the details of using such techniques transparent to the user.

1.3 Overview

This manual describes how to use the `cspc` compiler, to generate CSPs from higher level descriptions. In section 2, the first order language for the high level descriptions, is defined and explained. Section 3 describes how the language can be used to define CSPs, and finally section 4 contains details on how to invoke `cspc` and other programs associated with it. The appendices contain various technical information, description of output languages, details on how to extend the compiler, and more.

2 The Problem Description Language

2.1 Introduction

The language used to describe constraint satisfaction problems in `cspc` is a sorted first order language with finite sorts. It has sorts, variables, constants, functions, predicates, logical variables and all the first order logic connectives and quantifiers. This basic language is strong enough to describe any constraint satisfaction problem.

To make it easier to write problem descriptions, and make the eventual CSP smaller in size, there are a number of additional declarations and commands. These allow the user to define their own set of variable names, define functions and relations, attach procedures to functions and relations and to use extension procedures in the solver.

2.2 Declarations

2.2.1 Introduction

As the language is strongly typed, each sort, function and predicate must be declared at some point in the problem description. Although there is almost full freedom in the order used within the description, it is required that any sort declaration comes before the sort is used for the first time in another declaration. Declarations are characterized by capitalized keywords that describe what type of declaration is to follow.

2.2.2 Sorts

Sorts can be declared in two ways, either as a set of named symbols or as a range $[0, \ldots, n]$ of integers. The first method is used for sorts that have named members, such as a sort of colors:

```
SORT color = {red,yellow,green,blue,white}
```

The second method is used to declare a set of numbers, such as situations and times:

```
SORT time[50]
```

which declares the sort `time` to be the set $\{0, 1, \ldots, 49\}$.

2.2.3 Predicates and Functions

Predicates and functions are declared by specifying the sort of each argument, and the range for functions. For predicates there is a specific declaration of the form:

```
PRED <pred-name>(<arg-sort-1>,<arg-sort-2>,...,<arg-sort-k>)
```

Example:

```
PRED connected(node,node)
```

declares `connected` to be a relation on pairs of nodes.

For functions the declaration is in similar form:

```
FUNC <func-name>(<arg-sort-1>,<arg-sort-2>,...,<arg-sort-k>) : <range-sort>
```

Another example:

```
FUNC colorOf(country):color
```

declares that `colorOf` maps a country to a color.

Note that the domain of a function can be empty, i.e. `k=0`, making that function a constant function, but domains for predicates cannot be empty, requiring `k>0`.

2.2.4 Variable Names

Later we will get to variables and how they are used, at the moment we are just concerned with the names used for them. By default, a logical variable is recognized by its name starting with a question mark `?`. But a specific set of logical variable names can be declared in the description, allowing names that do not start with a question mark to be used as variable names. This is done using the variable declaration which is of the form:

```
VARS = {<name-1>,<name-2>,...,<name-k>}
```

As an example, if we want to use `x`, `y` and `z` for logical variables, we declare:

```
VARS = {x,y,z}
```

2.3 Expressions and Formulas

2.3.1 Introduction

Now that each of the objects that are used in the language have been declared, let us turn to using them to describe a problem. We will describe this main part of the language, by gradually building up the more complex formulas that can be used.

Note that the building blocks of the language can be divided into two disjoint sets, expressions and formulas. Expressions always have a value in some specific sort, while formulas take on truth values, they are either true or false. There are only three types of expressions: constants, variables and function calls. Everything else in is a formula.

2.3.2 Constants

Constants are either integers or named members of sorts. When a constant is found, the type of the corresponding parameter is checked. The following rules govern the typing and accepting of constants:

- If the constant is an integer value, and the sort is an integer range, the constant is accepted (as long as it is within range).
- If the constant is an integer value, and the sort is a set sort, the constant is accepted, but a warning is printed.
- If the constant is a named constant, it is accepted if and only if the sort is a set sort, and has this constant as member.

2.3.3 Variables

Logical variables are represented by variable names starting with a question mark, or names that have been declared to be logical variable names. This is done to distinguish variables from constants, since a variable and a constant with the same name can cause serious ambiguity.

Within a formula, variables are used in the same way as constants, the only difference is how the type checking is done. Each variable has a scope, which will be explained fully when we get to quantifiers. For the time being, let us just say that the scope of variable x is some formula. Each occurrence of the variable, within that formula, must be of the same sort, where the sort is determined by the defined parameter of the function or predicate where the variable appears.

2.3.4 Function Terms

Given a function that has been declared at some point in the description, a function term for that function is of the form:

```
<func-name>(<arg-1>,...<arg-k>)
```

where the number of arguments matches the number of arguments in the declaration, and each argument ¡arg-i¿ is an expression that has a sort that matches the corresponding parameter sort in the function declaration.

As an example, let us look at the function we declared earlier as:

```
FUNC colorOf(country):color
```

Then,

```
colorOf(?x)
colorOf(Holland)
colorOf(countryNumber(3))
```

are valid function terms, assuming variable `?x`, constant `Holland` and `countryNumber(3)` are of type `country`.

2.3.5 Predicates

Given a predicate that has been declared, an atom is of the form:

```
<pred-name>(<arg-1>,...<arg-k>)
```

where the number of arguments matches the number of arguments in the declaration, and each argument ¡arg-i¿ is an expression that has a sort that matches the corresponding parameter sort in the predicate declaration.

As an example, let us look at a small predicate:

```
PRED isRed(country)
```

Then,

```
isRed(?x)
isRed(Holland)
isRed(countryNumber(3))
```

are valid atoms, assuming variable `?x`, constant `Holland` and `countryNumber(3)` are of type `country`.

2.3.6 Logical Operators

The language has all the standard logical operators, negation, conjunction, disjunction, implications, equivalence and quantifiers.

Negation is represented by a unary minus sign (-), a tilde ( ) or the word `NOT`. Given a valid formula `<form>`:

```
-<form>
~ <form>
NOT <form>
```

are all valid formulas.

Conjunction is represented by a multiplication sign (*) or the word AND. .Given two valid formulas <form-1> and <form-2>:

<form-1> * <form-2>
<form-1> AND <form-2>

are also valid formulas.

Disjunction is represented by an add sign (+) or the word OR. Given two valid formulas <form-1> and <form-2>:

<form-1> + <form-2>
<form-1> OR <form-2>

are also valid formulas.

Implication is represented by a simple right arrow, made of a minus sign and a greater than sign (->), the word IMPLIES, or the inverse of "if" FI. Given two valid formulas <form-1> and <form-2>:

<form-1> -> <form-2>
<form-1> IMPLIES <form-2>
<form-1> FI <form-2>

are also valid formulas.

Reverse implication is represented by a simple left arrow, made of a a less than sign and a minus sign (<-), or the word IF. Given two valid formulas <form-1> and <form-2>:

<form-1> <- <form-2>
<form-1> IF <form-2>

are also valid formulas.

Equivalence is represented by a simple left and right arrow, made of a a less than sign, a minus sign and a greater than sign, (<->), or the word IFF. Given two valid formulas <form-1> and <form-2>:

<form-1> <-> <form-2>
<form-1> IFF <form-2>

are also valid formulas.

Universal quantification is represented by a the letter A, or the word FORALL. Given a valid formula <form>, and a variable name <var>:

A ¡var¿ ¡form¿
FORALL ¡var¿ ¡form¿

are also valid formulas.

Existential quantification is represented by a the letter E, or the word EXISTS. Given a valid formula <form>, and a variable name <var>:

E ¡var¿ ¡form¿
EXISTS ¡var¿ ¡form¿

are also valid formulas.

All operators are left associative, and are in the following order of priorities, highest first:

NOT, FORALL, EXISTS
AND
OR
IF,IMPLIES
IFF

It should be noted that the high precedence of quantifiers means that each applies to the smallest possible subformula that follows it. Thus:

```
A x p(x) -> q(x)
```

is parsed as:

```
(A x p(x)) -> q(x)
```

which is equivalent to:

```
(A x p(x)) -> q(y)
```

2.3.7 Variable Scope

When variables were described, the exact definition of the scope of a variable, was missing. Now that we have described quantifiers we can fix that. The scope of a variable is the smallest subformula that starts with a quantifiers specifying this variable name, excluding any subsubformulas that start with a quantifier specifying the same name. To be a valid variable, the variable must match the same sort in each appearance in its entire scope.

In this language, the user can use variables without specifying the quantifier for it. In that case, the variable is viewed as being universally quantified over the whole formula it appears in. Thus

```
p(x) -> q(x)
```

is equivalent to:

```
A x (p(x) -> q(x))
```

2.4 Commands

2.4.1 Introduction

Although the language that has been described is enough to represent any CSP, more is needed to make it easy to use and efficient. This is done with additions to the compiler, called commands. Commands come in two flavors, there is a set of special purpose built-in commands, such as assignments, and there is a set of general commands, which can be extended by adding new commands to the compiler.

2.4.2 Defined Functions and Relations

As we will get to later, functions and predicates will correspond to variables in the constraint satisfaction problem. For example, the function `colorOf(country):color` will correspond to a set of variables assigning colors to countries in a map coloring problems. But there are certain functions and predicates that should not be viewed as variables, but rather as simple calculations. A simple example is the equality relation.

Typically, such relations and functions are handled by adding axioms to guarantee they will get the correct value. For equality, the typical axiomatization is:

```
eq(x,x)
eq(x,y) -> eq(y,x)
eq(x,y) * eq(y,z) -> eq(x,z)
```

But this is rather inefficient, a better method is to assign meaning to the `eq` relation, such that when both arguments have been instantiated, the value of the relation is simply calculated. In `cspc` the values of functions and predicates can be specified by the user.

To specify a relation, the command `%DefinedRelation` is used. Given a single predicate as an argument, this command notes that the relation is to be considered as a defined relation, not as a set of variables. The values for the relation are specified by using atomic formulas, specifying whether an instance is true or false. As an example, let us look at a very simple defined relation `isTrue`:

```
SORT color = {red,yellow,green}
PRED isGoColor(color)
%DefinedRelation(isTrue)
isGoColor(green)
-isGoColor(yellow)
-isGoColor(red)
```

When the relation `isGoColor` appears with other objects, the value is calculated directly as soon as the argument has been instantiated to a value. If that value is `red`, the predicate instance is replaced by the truth value `T`, but if the value is `yellow` or `green` it becomes false `F`.

Finally, let us note that in the specification here above, the last two lines `-isGoColor(yellow)` and `-isGoColor(red)` are not needed. Any instance of a defined relation, that is not specified, automatically evaluates to false.

To specify the values for a function, there are two methods, which can be used separately or in combination. The first is to specify values as part of the declaration of that function. This is done as follows:

```
SORT num[5]
FUNC half(num):num = { (0):0, (1):0, (2):1, (3):1, (4):2 }
```

Whenever the function `half` is encountered, the instance is evaluated as soon as possible. E.g. the expression:

```
eq(half(3),2)
```

is immediately replaced by:

```
eq(1,2)
```

The other method for specifying values for defined functions is to use assignments. The function `half` can be specified as follows:

```
SORT num[5]
FUNC half(num):num
half(0) = 0
half(1) = 0
half(2) = 1
half(3) = 1
half(4) = 2
```

and the meaning is exactly the same as the earlier definition.

The two methods can be used together, specifying some values in the declaration, and other values with assignment statements. There are three rules that should be remembered when using defined functions:

1. If a function is a part of an assignment, or has a value specification list after its declaration, it is considered defined and does not generate any variables in the CSP.
2. If a function is defined, but the value for a certain instance is not specified, the first value of the range sort is assigned as the value.
3. If the value of an instance is specified more than once, the latest specification is used.

2.4.3 Use of Extension Procedures

2.4.4 General Procedures

The other method for specifying values for relations and functions, is to use particular built-in procedures. Each procedure is an independent object, making it possible for a user to add procedures that are not yet there. To see how such procedures are used, let us look at an example:

```
SORT num[10]
PRED eq(num,num)
%EqualityRelation(eq)
```

This specifies that the predicate `eq` is to be an equality relation, using the equality procedure to calculate its value. Each time an instance of `eq` is encountered, where both arguments have been evaluated to single values, the procedure is called, and it will return true if and only if the two arguments are equal.

Currently, there are a number of procedures built into the compiler, and the list is bound to grow:

%AddFunction is a procedure that takes two values i and j and returns the sum $i + j$.

%SubFunction is a procedure that accepts two values i and j and returns $i - j$.

%MulFunction is a procedure that takes two values i and j and returns ij.

%IncFunction is a procedure that takes a single value i and returns $i + 1$.

%DecFunction is a procedure that takes a single value i and returns $i - 1$.

%EqualityRelation is a procedure that takes two values i and j and returns true if $i = j$, false otherwise.

%LessThanRelation is a procedure that takes two values i and j and returns true if $i < j$, false otherwise.

%GreaterThanRelation is a procedure that takes two values i and j and returns true if $i > j$, false otherwise.

3 Writing Problem Descriptions

3.1 Introduction

Now that we have described the language, it is time to explain how to use it to represent constraint satisfaction problems. The main issues in doing this are:

1. Representing the CSP variables and their domains.
2. Representing the constraints.
3. Defining and using relations and functions.
4. Efficiency considerations.

3.2 Basic Ideas

Let us introduce the basics by using an example, a small map coloring problem. We have four countries, Algeria, Burma, Canada and Denmark. Every country is adjacent to every other country, except that Denmark and Algeria don't touch. We have three colors, red,blue and green to color the countries, such that no adjacent countries are colored with the same color.

3.2.1 Representing the variables and their domains

It is possible to directly represent the CSP using this language, just by isolating variables, domains and constraints. In this case the variables are the color of each country, and their domains are the set of colors. This can easily be represented as:

```
SORT color = {red,green,blue}
FUNC colorOfAlgeria():color
FUNC colorOfBurma():color
FUNC colorOfCanada():color
FUNC colorOfDenmark():color
```

Seeing the prefix "colorOf" on each of the functions leads to a more compact (and perhaps also more intuitive) representation, by viewing the variables as function evaluations. Then we have two sets of items in the universe, colors and countries, and a map from each country to a color:

```
SORT color = {red,green,blue}
SORT country = {Algeria, Burma, Canada, Denmark}
FUNC colorOf(country):color
```

We will adapt the second representation as it is more compact and more interesting.

3.2.2 Representing the constraints

The only constraint in this problem is that for any two adjacent countries, the function `colorOf` maps them to different colors. To be able to represent this constraint we need to be able to indicate which countries share borders, and which colors are the same. So we define:

```
PRED sameColor(color,color)
PRED adjacent(country,country)
```

To represent which countries are adjacent, we could list the whole relation, but we can save a bit in writing by noting that the adjacency relation is symmetric. Then we can write:

```
adjacent(Algeria, Burma)
adjacent(Algeria, Canada)
adjacent(Burma,Canada)
adjacent(Burma,Denmark)
adjacent(Canada,Denmark)
adjacent(x,y) -> adjacent(y,x)
```

Finally, we can write the main constraint:

```
adjacent(x,y) -> -(sameColor(colorOf(x),colorOf(y)))
```

3.2.3 Defining and using relations and functions

Now that we have the variables, domains and the constraints, we need to write down axioms for the predicates and functions used in the constraints. Compiling and solving what we have gotten so far would most likely result in a solution where no two colors are the same, not even `red` and `red`. So we need to say what the `sameColor` and `adjacent` have to satisfy.

To represent an equality relation, the simplest method is to write up two of the three axioms, and then indicate every pair of items that is not equal:

```
sameColor(x,x)
sameColor(x,y) -> sameColor(y,x)
-sameColor(red,blue)
-sameColor(red,green)
-sameColor(blue,green)
```

And even though we can get away with some laziness in this problem, we should also give similar axioms for the adjacency relation:

```
-adjacent(x,x)
-adjacent(Algeria,Denmark)
```

to make sure we get exactly the relation we want.

3.2.4 Efficiency Considerations

It seems strange to use half the formulas written to represent the definition of an equality relation on a set, when we know exactly what the relation is, and don't really need to prove it logically every time we solve a problem like this.

So, instead of writing all the axioms for equality, we use a compiler command to indicate that `sameColor` is an equality relation. This is done by replacing the axioms with:

```
%EqualityRelation(sameColor)
```

The result of this command is that the compiler defines the predicate by a procedural attachment. In the case of translating the problem to a SAT problem, this means that for any ground instance of `sameColor` the compiler calls the procedural attachment to evaluate it, and then depending on the whether it evaluates to true or false, removes the clause it appears in or removes the literal, respectively.

There are more commands available, all of which are described further here below. But even with a large collection of such commands, it is possible that there is a function with a well known mapping, that is not available as a procedural attachment. To allow such functions to be computed as efficiently, the compiler can accept function definition statements in the database, and as part of the definition of the function. An example of such a function is rotation for directions:

```
SORT direction = {north,east,south,west}
FUNC rotate(direction):direction =
    {(north):east, (east):south, (south):west, (west):north}
```

An equivalent definition where the value assignments show up as part of the database, would be:

```
SORT direction = {north,east,south,west}
FUNC rotate(direction):direction
....
rotate(north) = east
rotate(east) = south
rotate(south) = west
rotate(west) = north
```

3.2.5 Compiling and Solving

Let us assume we have the theory given above (excluding the command) in a file called `mapcol.fol`. To compile it we give the following command:

```
cspc -d mapcol.dis mapcol.fol mapcol.bin
```

which creates two new files. The file `mapcol.bin` contains a SAT representation of the theory, while `mapcol.dis` contains the information needed to translate the model back into the first order language.

To solve the problem, with `wsat` for an example, we do:

```
wsat mapcol.bin
```

This will give us a model, which we can put into a file `mapcol.sol`, and then get a readable model by:

```
model2sym mapcol.dis mapcol.sol
```

Please note that the model accepted by the translator `model2sym` must be a list of integers representing members of the model, or a negative number representing different reasons for failing to find a solution. What we have so far is that -1 represents that the theory was proven to be unsatisfiable and that -2 indicates that the solver gave up before finding a solution.

4 Using the cspc Compiler

4.1 Invoking the Compiler

The syntax for invoking the compiler is:

```
cspc [options] [ [input-file-name] output-file-name]
```

If the name of the output file is omitted, the results are sent to the standard output. If both file names are omitted, standard I/O is used for both input and output.

The following options can be supplied to control the compilation:

-1 makes single letter symbols stand for logical variables. This is done for backwards compatibility reasons.

-c specifies that the resulting output should be a standardized CSP format, which is basically a set of relationalized clauses, where logical variables define a set of CSP variables, and function result values are viewed as CSP values.

-d file-name specifies that information for the decompiler should be written to the given file. This information is used by the translator `model2sym` to interpret the given model in terms of the predicates and functions described in the original input file. This option is mainly useful for using satisfiability engines that are not provided with this compiler.

-e [level] means that the compiler will calculate an estimate for the number of clauses produced. The optional level argument determines how detailed the calculation and reports are. Level 1, which is the default means that only a complete estimate is given, while level 2 gives the estimates for each clause and function definition.

-s means skipping the Skolem functions when writing the translation table. Done to make translations smaller in cases where the results of the Skolem functions are not interesting.

-t indicates that clauses should be written as comments in the boolean SAT file. Note that some SAT solvers cannot handle comments in the problem description.

-v makes the compiler run in verbose mode, giving status reports as it runs. These reports are sent to standard error so redirection of standard output is not affected by this.

4.2 Using the Model Interpreter

To use the model interpreter to decode the model given by SAT solvers, the problem should be compiled using the `-d` option. The model interpreter is invoked by:

```
model2sym [-c] <trans-file> [<model-file>]
```

The `trans-file` is the file specified with the `-d` option for `cspc`, and the `model-file` contains a list of integers, each representing a member of the model found (if one was found). If no model was found, the `model-file` should start with a negative integer, -1 if the theory was proved unsatisfiable, and -2 if the solver gave up before finding a solution.

The `-c` option means the translation file and the model file are based on CSPs, not on SAT problems. Interpretation is done accordingly.

5 Appendix

5.1 Input format

database	→	decl database
	∨	formula database
	∨	% command database
	∨	assignment database
	∨	λ
decl	→	vars-decl
	∨	sort-decl
	∨	pred-decl
	∨	func-decl
	∨	func-decl = { func-value-list }
vars-decl	→	vars-def-op = { var-name-list }
sort-decl	→	sort-def-op *sort-name* = { item-list }
	∨	sort-def-op *sort-name* [*int-value*]
	∨	sort-def-op *sort-name* = [*sort-list*]
vars-def-op	→	**VARS**
	∨	**VAR**
	∨	**DEFVARS**
	∨	**LOGVARS**
sort-def-op	→	**SORT**
	∨	**SORTDEF**
item-list	→	item-list , *item-name*
	∨	*item-name*
pred-decl	→	pred-def-op *pred-name* (sort-list)
pred-def-op	→	**PRED**
	∨	**PREDDEF**
func-decl	→	func-def-op *func-name* ( ) : *sort-name*
	∨	func-def-op *func-name* (sort-list) : *sort-name*
func-def-op	→	**FUNC**
	∨	**FUNCDEF**
sort-list	→	sort-list , *sort-name*
	∨	*sort-name*
func-value-list	→	func-value func-value-list
	∨	func-value

func-value	→	(value-list) = value
value-list	→	value , value-list
	∨	λ
value	→	*int-value*
	∨	*item-name*
formula	→	forall-op *var-name* formula
	∨	exists-op *var-name* formula
	∨	formula and-op formula
	∨	formula or-op formula
	∨	formula iff-op formula
	∨	formula if-op formula
	∨	formula implies-op formula
	∨	not-op formula
	∨	atom
forall-op	→	**A** | **FORALL**
exists-op	→	**E** | **EXISTS**
and-op	→	* | **AND**
or-op	→	+ | **OR**
not-op	→	− | ∼ | **NOT**
iff-op	→	<-> | **IFF**
if-op	→	<- | **IF**
implies-op	→	-> | **FI** | **IMPLIES**
atom	→	*pred-name* (term-list)
term-list	→	term-list , term
	∨	term
term	→	*func-name* (term-list)
	∨	*func-name* ( )
	∨	*int-value*
	∨	*item-name*
assignment	→	*func-name* (value-list) = value

5.2 Output Format

5.2.1 SAT Output Format

There are a few different types of format floating around. The simplest is the one we currently use, which is compatible with all CIRL implementations of tableau and wsat.

The language is defined as follows:

`theory`	→	`clause theory`
	∨	`clause % query`
`clause`	→	`( literal-list )`
`literal-list`	→	`literal literal-list`
	∨	`literal`
`literal`	→	*nat-number*
	∨	`-` *nat-number*
`query`	→	`literal`
	∨	**0**

where *nat-number* $\in \mathbb{N}$.

5.2.2 CSP Output Format

`problem`	→	`header body`
`header`	→	*nr-of-decls* `decl-list`
`decl-list`	→	`decl decl-list`
	∨	`decl`
`decl`	→	*function-number nr-of-args* `arg-size-list` *arg-size*
`arg-size-list`	→	*arg-size* `arg-size-list`
	∨	λ
`body`	→	*nr-of-constraints* `constraint-list`
`constraint-list`	→	`constraint constraint-list`
	∨	λ
`constraint`	→	*nr-of-atoms nr-of-vars* `atom-list`
`atom-list`	→	`atom atom-list`
	∨	`atom`
`atom`	→	*func-number* `lit-list lit`
`lit-list`	→	`lit list-list`
	∨	λ
`lit`	→	*constant-number*
	∨	$-$ *variable-number*

5.3 Adding Procedures

Earlier we described procedures that can be used within the compiler, such as addition, equality etc. It is relatively easy for a user to add new procedures, that can then be used just like those built in.

To add a new procedure, edit the file NewProcs.hh which comes with the distribution.

Appendix E

Symmetry-Breaking Predicates for Search Problems*

James Crawford
Matthew Ginsberg
Computational Intelligence Research Laboratory
1269 University of Oregon
Eugene, OR 97403-1269
{jc, ginsberg}@cirl.uoregon.edu

Eugene Luks
Amitabha Roy
Department of Computer Science
The University of Oregon
Eugene, OR 97403-1202
{luks, aroy}@cs.uoregon.edu

February 4, 1999

Abstract

Many reasoning and optimization problems exhibit symmetries. Previous work has shown how special purpose algorithms can make use of these symmetries to simplify reasoning. We present a general scheme whereby symmetries are exploited by adding "symmetry-breaking" predicates to the theory. Our approach can be used on any propositional satisfiability problem, and can be used as a pre-processor to any (systematic or non-systematic) reasoning method. In the general case adding symmetry-breaking axioms appears to be intractable. We discuss methods for generating partial symmetry-breaking predicates, and show that in several specific cases symmetries can be broken either fully are partially using a polynomial number of predicates. These ideas have been implemented and we include experimental results on two classes of constraint-satisfaction problems.

1 Introduction

Human artifacts from chess boards to aircraft exhibit symmetries. From the highly regular patterns of circuitry on a microchip, to the interchangeable pistons in a car engine, or seats in a commercial aircraft, we are drawn esthetically and organizationally to symmetric designs. Part of the appeal of a regular or symmetric design is that it allows us to reason about and understand larger and more complex structures than we could otherwise handle. It follows that if we are to build computer systems that configure, schedule, diagnose, or

*This paper appeared in *Proc. KR'96.*

otherwise reason about human artifacts, we need to endow these reasoning systems with the ability to exploit structure in general and symmetries in particular.

Automated reasoning is a huge area, so for purposes of this paper we will focus on search. Abstractly, a search problem consists of a large (usually exponentially large) collection of possibilities, the *search space*, and a predicate. The task of the search algorithm is to find a point in the search space that satisfies the predicate. Search problems arise naturally in many areas of Artificial Intelligence, Operations Research, and Mathematics.

The use of symmetries in search problems is conceptually simple. If several points in the search-space are related by a symmetry then we never want to visit more than one of them. In order to accomplish this we must solve two problems. First, the symmetries need to be discovered; e.g., we need to realize that we can interchange the five ships without changing the basic form of the problem. Second, we need to make use of the symmetries.

This paper focuses on the second of these problems. It was shown by Crawford [8] that detecting symmetries is equivalent to the problem of testing graph isomorphism, a problem that has received a substantial amount of study (see, *e.g.*, [1]).

With regard to taking computational advantage of the symmetries, past work has focused on specialized search algorithms that are guaranteed to examine only a single member of each symmetry class [5, 8]. Unfortunately, this makes it difficult to combine symmetry exploitation with other work in satisfiability or constraint satisfaction, such as flexible backtracking schemes [13, 14] or nonsystematic approaches [22, 24]. Given the rapid progress in search techniques generally over the past few years, tying symmetry exploitation to a specific search algorithm seems premature.

The approach we take here is different. Rather than modifying the search algorithm to use symmetries, we will use symmetries to modify (and hopefully simplify) the problem being solved. In tic-tac-toe, for example, we can require that the first move be in the middle, the upper left hand corner, or the upper middle (since doing this will not change our analysis of the game in any interesting way). In general, our approach will be to add additional constraints, *symmetry-breaking predicates*, that are satisfied by exactly one member of each set of symmetric points in the search space. Since these constraints will be in the same language as the original problem (propositional satisfiability for purposes of this paper) we can run the symmetry detection and utilization algorithm as a preprocessor to any satisfiability checking algorithm.

Of course there is a catch. In this case two catches: First, there is no known polynomial algorithm for detecting the symmetries. Symmetry detection is equivalent to graph isomorphism which is believed to be easier than NP-complete, but is not known to be polynomial. Nevertheless, graph isomorphism is rarely difficult in practice, as has been profoundly demonstrated by the efficient nauty system [21]. Furthermore, it has been shown that, on average, graph isomorphism is in linear time using even naive methods [2]. The second catch is that even after detection is complete, computing the full symmetry-breaking predicate appears to be intractable.

However, there is generally no reason to generate the full symmetry breaking predicate. We can generate a partial symmetry-breaking predicate without affecting the soundness or completeness of the subsequent search. We will show that in several interesting cases we can break symmetries either fully or partially using a polynomial number of predicates.

The outline of the rest of this paper is as follows: we first define symmetries of search problems, and discuss how predicates can be added to break symmetries. We then discuss both exact and partial methods for controlling the size of the symmetry-breaking predicate. Finally we discuss experimental results and related work.

2 Definitions and Preliminaries

For purposes of this paper we will assume that we are working in clausal propositional logic. The symmetries of a propositional theory will be defined to be the permutations of the variables in the theory that leave the theory unchanged. These symmetries form a group and we use techniques and notation from computational group theory throughout the paper.

Let L be a set of propositional variables. As usual, literals are variables in L, or negations of variables in L. If $x \in L$, then we write the negation of x as $\overline{x}$. A clause is then just a disjunction of literals (written, *e.g.*, $x \vee y \vee z$), and a theory is a conjunction of clauses. One basic observation that will be critical to the definitions below is that two clauses are considered to be identical iff they involve the same set of literals (*i.e.*, order is not significant) and two theories are identical iff they involve the same set of clauses.

A *truth assignment* for a set of variables L is a function $A : L \to \{t, f\}$ (on occasion we write $1, 0$ for t, f respectively). In the usual way, A extends by the semantics of propositional logic to a function on the set of theories over L, and, by abuse of notation, we continue to denote the extended function by A.[1] A truth assignment A of L is called a *model* of the theory T if $A(T) = t$. The set of models of T is denoted $\mathcal{M}(T)$.

The propositional satisfiability problem is then just (see, *e.g.*, [12]):

Instance: A theory T.

Question: Is $\mathcal{M}(T)$ non-empty (*i.e.*, does T have a model)?

Clearly one can determine whether such an assignment exists by trying all possible assignments. Unfortunately, if the set L is of size n then there are 2^n such assignments. All known approaches to determining propositional satisfiability are computationally equivalent (in the asymptotically worst case) to such a complete search. Propositional satisfiability is thus one of the simplest "canonical" examples of a search problem.

To formally define symmetries we need some additional notation. Consider a set L. The group of all permutations of L is denoted by $\mathrm{Sym}(L)$.[2] This is a group under composition; the product $\theta\phi$ of $\theta, \phi \in \mathrm{Sym}(L)$ is taken to be the result of performing θ and then ϕ. If $v \in L$ and $\theta \in \mathrm{Sym}(L)$, the image of v under θ is denoted v^θ (it is standard to write the permutation as a superscript so that we can make use of the natural equality $v^{\theta\phi} = (v^\theta)^\phi$). A permutation θ of a set L of variables naturally extends to a permutation of negated variables such that $\overline{v}^\theta = \overline{v^\theta}$, and thus to a permutation of the set of clauses over L, wherein

[1] That is, $A(\overline{x})$ is the negation of $A(x)$, A is true of a clause iff it is true of at least one of the terms in the clause, and A is true of a theory iff it is true of all the clauses in the theory.

[2] Recall that a permutation of a finite set L is a one-to-one mapping $\theta : L \to L$.

if $C = \bigvee_{i=1}^{r} v_i$, then $C^\theta = \bigvee_{i=1}^{r} v_i^\theta$, and finally to a permutation of the theories over L, namely, if $T = \{C_i\}_{1 \le i \le m}$, then $T^\theta = \{C_i^\theta\}_{1 \le i \le m}$.

Let T be a theory over L and let $\theta \in \mathrm{Sym}(L)$. We say that θ is a *symmetry*, or automorphism, of L iff $T^\theta = T$. The set of symmetries of T is a subgroup of $\mathrm{Sym}(L)$ and is denoted by $\mathrm{Aut}(T)$.

For example, consider the following theory: $a \vee \bar{c}$, $b \vee \bar{c}$, $a \vee b \vee c$, $\bar{a} \vee \bar{b}$. Notice that if we interchange a and b the theory is unchanged (again, only the order of the clauses and the order of literals within the clauses is affected). It is customary to denote this particular symmetry by $(a\ b)$.

Permutations of variables in general, and symmetries in particular, can be viewed as acting on assignments as well as theories. If $\theta \in \mathrm{Sym}(L)$ then θ acts on the set of truth assignments by mapping $A \mapsto {}^\theta A$, where ${}^\theta A(v) = A(v^\theta)$ for $v \in L$.[3] Hence, if T is a theory over L, $A(T^\theta) = {}^\theta A(T)$. Thus, we have the immediate consequence that any symmetry of T maps models of T to models of T, and non-models of T to non-models:

Proposition 2.1 *Let T be a theory over L, $\theta \in \mathrm{Aut}(T)$, and A a truth assignment of L. Then $A \in \mathcal{M}(T)$ iff ${}^\theta A \in \mathcal{M}(T)$.*

More generally, $\mathrm{Aut}(T)$ induces an equivalence relation on the set of truth assignments of L, wherein A is equivalent to B if $B = {}^\theta A$ for some $\theta \in \mathrm{Aut}(T)$; thus, the equivalence classes are precisely the *orbits* of $\mathrm{Aut}(T)$ in the set of assignments. Note, further, that any equivalence class either contains only models of T, or contains no models of T. This indicates why symmetries can be used to reduce search: we can determine whether T has a model by visiting each equivalence class rather than visiting each truth assignment.

3 Symmetry-Breaking Predicates

The symmetry-breaking predicates are chosen such that they are true of exactly one element in each of the equivalence classes of assignments generated by the symmetry equivalence. For example, for the small example theory discussed in section 2, the two models are (t, f, f) and (f, t, f). The theory has one non-trivial symmetry – the interchange of a and b. As required by proposition 2.1, applying this perturbation to a model yields a model. We can "break" the symmetry by adding the axiom $a \rightarrow b$ which eliminates one of the models, (t, f, f), leaving us with only one model from the equivalence class.

In general, we introduce an ordering on the set of variables, and use it to construct a lexicographic order on the set of assignments. We will then add predicates that are true of only the smallest model, under this ordering, within each equivalence class.[4] Intuitively we do this by viewing each model as a binary number (*e.g.*, (t, f, f) would be seen as 100). We then add predicates saying that θ does not map M to a smaller model (for all symmetries θ).

[3] It is natural to write this as a "left action," e.g., we have ${}^{\theta\phi}A = {}^\theta({}^\phi A)$, whereas expressing the image of A under θ by A^θ would lead to the awkward relation $A^{\theta\phi} = (A^\phi)^\theta$.

[4] We note that this is surely not the *only* way to create symmetry-breaking predicates. One can break symmetries by adding any predicate that is true of one member of each equivalence class.

If $\theta \in \mathrm{Sym}(L)$, then θ acts on any sequence of variables in L: if $V = (v_1, \ldots, v_m)$ then $V^\theta = (v_1^\theta, \ldots, v_m^\theta)$. For a sequence $V = (v_1, \ldots, v_m)$ and $0 \leq i \leq m$, it is convenient to denote by V_i the initial segment $(v_1, \ldots, v_i)$ (of course, this is the empty sequence () if $i = 0$).

If $v, w \in L$, we write $v \leq w$ as a shorthand for the clause $v \to w$. If $V = (v_1, \ldots, v_m)$ and $W = (w_1, \ldots, w_m)$ are sequences of variables in L and $0 \leq i \leq m$, we let $P_i(V, W)$ abbreviate the predicate

$$V_{i-1} = W_{i-1} \to v_i \leq w_i.$$

Finally, we write $V \leq W$ as shorthand for

$$\bigwedge_{i=1}^{m} P_i(V, W),$$

that is,

$$\begin{aligned} v_1 \leq w_1 \wedge \\ (v_1 = w_1) \to v_2 \leq w_2 \wedge \\ (v_1 = w_1 \wedge v_2 = w_2) \to v_3 \leq w_3 \wedge \\ \ldots \end{aligned}$$

The intuition behind this definition is that if we have an assignment A, then the predicate $V \leq W$ will be true of A iff $A(V) = (A(v_1), \ldots, A(v_m))$, viewed as a binary number, is less than or equal to $A(W) = (A(w_1), \ldots, A(w_m))$.

Henceforth, we fix an ordering $V = (v_1, \ldots, v_m)$ of the variables in L. Then the set of truth assignments of L inherit a lexicographic ordering, i.e., $A < B$ if, for some i, $A(v_j) = B(v_j)$ for $j < i$, while $A(v_i) < B(v_i)$. In other words, viewed as binary numbers, $A(V) < B(V)$.

Now consider a symmetry θ of a theory T. The predicate $V \leq V^\theta$ rules out any model M for which $M > {}^\theta M$. It is immediate that

Proposition 3.1 *Let T be a theory, and V be an ordering of it's variables. Then the predicate*

$$\bigwedge_{\theta \in \mathrm{Aut}(T)} V \leq V^\theta$$

is true only of the lexicographically least model in each equivalence class of truth assignments. Hence, it is a symmetry-breaking predicate for T.

Returning to the example we have been tracking, we take $V = (a, b, c)$. Recall that θ swaps a and b. Thus $V^\theta = (b, a, c)$, so $V \leq V^\theta$ is:

$$\begin{aligned} a \to b \\ a = b \to (b \to a) \\ (a = b \wedge b = a) \to (c \to c) \end{aligned}$$

In which the only non-tautologous term is $a \rightarrow b$. This rules out the model (t, f, f). This model is ruled out because θ maps it to the lexicographically smaller model (f, t, f).

By addition of auxiliary variables the predicates given by $V \leq V^\theta$ can be represented by a linear number of clauses. We do this by introducing a new variable e_i defined to be true exactly when $v_i = v_i^\theta$. This can be done by adding the following clauses:

$$(v_i \wedge v_i^\theta) \rightarrow e_i$$
$$(\overline{v_i} \wedge \overline{v_i^\theta}) \rightarrow e_i$$
$$(e_i \wedge v_i) \rightarrow v_i^\theta$$
$$(e_i \wedge \overline{v_i}) \rightarrow \overline{v_i^\theta}$$

Nevertheless, since $\mathrm{Aut}(T)$ may be of exponential size, the entire symmetry-breaking predicate given by this theorem may be quite large. In general we have the following negative result:

Theorem 3.2 *The problem of computing, for any theory T, a predicate true of only the lexicographic leader in each equivalence class of models is NP-hard.*

The proof of this theorem is technical and is given in the appendix. The proof includes showing the NP-completeness of the following question: Given an incidence matrix A of a graph Γ, can one reorder the vertices and edges of Γ so as to produce an incidence matrix B that exceeds A lexicographically?

Despite this negative worst-case result, it is still possible to generate, either exactly or approximately, symmetry-breaking predicates for interesting problems. In the next two sections we focus on exact methods and show that in some cases where $\mathrm{Aut}(T)$ is exponential it may still be possible to generate tractable symmetry-breaking predicates. Then, in section 6, we turn to approximate symmetry breaking.

4 The Symmetry Tree

For problems, like n-queens, with a relatively small number of symmetries we are done: one simply computes the symmetries and then calculates the predicate for each symmetry. However, many interesting problems have many symmetries, and computing the predicates for each symmetry yields unnecessary duplication. For example, if $\theta, \phi \in \mathrm{Aut}(T)$ agree on the first i variables then $P_i(V, V^\theta) = P_i(V, V^\phi)$. In order to attack problems with a large number of symmetries, we first organize the symmetries into a *symmetry tree*, and then show how the tree can be "pruned". To describe the symmetry tree and pruning methods we need some notation for permutations.

Again, let T be a theory over L and $V = (v_1, \ldots, v_m)$ be a fixed ordering of L. We can describe a permutation of the variables by listing the image of V under the permutation. For example, the permutation taking v_1 to v_2, v_2 to v_3, and v_3 to v_1 can be written as $[v_2, v_3, v_1]$. The notation is extended to *partial permutations*, which are 1-1 maps of initial segments of V into V, thus the partial permutation taking v_1 to v_3 and v_2 to v_1 is written $[v_3, v_1]$. Note that initial segment could be empty, giving rise to the partial permutation $[\,]$.

For purposes of the formal development to follow, it is useful to describe these partial permutations with a standard group-theoretic construction. Let $G = \mathrm{Aut}(T)$. For $0 \leq i \leq n$, let G_i be the set of permutations in G that do not move the first i variables. That is, $G_i = \{\theta \in G \mid v_j^\theta = v_j, \text{ for } 1 \leq j \leq i\}$. Thus,

$$G = G_0 \supseteq G_1 \supseteq \cdots \supseteq G_n = 1$$

(the last being the identity subgroup). For $0 \leq i \leq n$, let $\mathcal{C}_i$ denote the set of *right cosets* of G_i in G. A right coset $C \in \mathcal{C}_i$ is a set that is of the form $G_i\theta$ for some $\theta \in G$ (note that G is the disjoint union $\bigcup_{C \in \mathcal{C}_i} C$). For purposes of this paper, one can think of a right coset as a partial permutation. For this, let $\theta \in C$, then the partial permutation of length i associated with θ is $[v_1^\theta, \ldots, v_i^\theta]$. Note that this i-tuple is independent of the choice of $\theta \in C$.

We can now describe the structure of the symmetry tree, $\mathrm{SB}(T)$, for T. The root of $\mathrm{SB}(T)$, considered to be at level 0, is G. The set $\mathcal{C}_i$ comprises the nodes at level i. Furthermore, $C \in \mathcal{C}_i$ is a parent of $C' \in \mathcal{C}_{i+1}$ iff $C' \subseteq C$. Equivalently, in terms of partial permutations, the root is [ ] and each node $[w_1, \ldots, w_{i-1}]$ will have one child $[w_1, \ldots, w_{i-1}, x]$ for each x that is the image of v_i under a symmetry mapping V_{i-1} to $(w_1, \ldots, w_{i-1})$.

To illustrate this construction, recall the example discussed in section 2. Assume that $V = a, b, c$. There are two symmetries of this theory: the identity operation, and the exchange of a and b. The first of these takes a to itself, and the second takes a to b. There will thus be two children of the root node: $[a]$ and $[b]$. Given that a is mapped to a, b is forced to map to b, so the node $[a]$ has only the child $[a\ b]$. Similarly the node $[b]$ has only the child $[b\ a]$. The final symmetry tree is shown in figure 1.

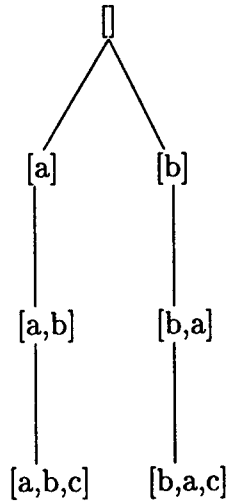


Figure 1: Symmetry trees for simple example.

The duplication previously observed in the symmetry-breaking predicate of Proposition 3.1 arose precisely because $P_i(V, V^\theta) = P_i(V, V^\phi)$ whenever θ and ϕ belong to the same

right coset of G_i. With this in mind, for $C \in \mathcal{C}_i$, we define $Q(C,i)$ to be $P_i(V,V^\theta)$ for any (all) $\theta \in C$. (The "i" in the notation "$Q(C,i)$" is not superfluous, that is, it is not determined by C; it is possible that $G_i = G_j$ for $j \neq i$ in which case $C \in \mathcal{C}_j(G)$.) Finally, we associate the predicate $Q(C,i)$ to the corresponding node $C \in \mathcal{C}_i$ in $\mathrm{SB}(T)$. It is now clear that the conjunction of the predicates assigned to the nodes of $\mathrm{SB}(T)$

$$\bigwedge_{i=1}^{n} \bigwedge_{C \in \mathcal{C}_i} Q(C,i)$$

remains a symmetry-breaking predicate for T.

5 Pruning the Symmetry Tree

Working from the symmetry tree to generate symmetry-breaking predicates eliminates a certain amount of duplication. However, there are cases in which the symmetry tree is of exponential size. For example, the theory $(x \vee y \vee z) \wedge (\overline{x} \vee \overline{y} \vee \overline{z})$ admits all $3! = 6$ permutations of $\{x,y,z\}$. Although, we do not typically expect to see all $n!$ permutations of the variables appearing in practical problems, it is not unusual to see theories with exponentially large symmetry groups. Furthermore, we would surely want to take advantage of the symmetry-breaking opportunities afforded by such a group.

In this section we show that pruning rules can achieve a drastic reduction in size of the symmetry tree in some important cases while still breaking all the symmetries. To see how this is done consider the symmetry tree shown in figure 2. Here, we suppose that $\mathrm{Aut}(T)$ includes the permutation $(x_1\ x_i)$, exchanging x_1 and x_i. Then, for any nodes in the symmetry of T of the form $[x_1, x_j]$ and $[x_i, x_j]$, $j \neq 1, i$, the subtree rooted at $[x_i, x_j]$ (namely, the tree T_2 in the diagram below) can be pruned.

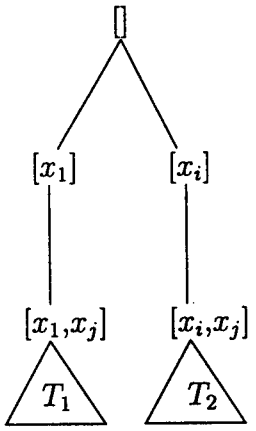


Figure 2: Symmetry trees for pruning example.

To see why this is so, consider any node $[x_i, x_j, \ldots, x_k]$ in T_2. Composing the symmetry creating this branch with $(x_1\ x_i)$, we must find a corresponding node $[x_1, x_j, \ldots, x_k]$ in T_1. In this example, it is not difficult to show that $Q([x_1, x_j, \ldots], k) \rightarrow Q([x_i, x_j, \ldots], k)$. Thus we can prune T_2 without loss of inferential power.

The example can be generalized as follows.

Consider a right-coset $C \in \mathcal{C}_i$. Take $\theta \in C$ (the constructions to follow are independent of the choice of θ). Let $\sim_{C,i}$ be the smallest equivalence relation on L such that $v_j \sim_{C,i} v_j^\theta$ for $1 \leq j \leq i$, i.e., $v \sim_{C,i} w$ if v and w are the endpoints of a sequence $u, u^\theta, u^{\theta\theta}, u^{\theta\theta\theta}, \ldots$ such that, with the possible exception of one of these endpoints, all terms are in V_i. Observe that, if w is in neither of the sequences V_i nor V_i^θ, then the $\sim_{C,i}$ equivalence class of w is a singleton.

From the definition of Q, one can see that if $x \sim_{C,i} y$ then for any child $C', i+1$ of C, i in $SB(T)$, the antecedent of $Q(C', i+1)$ forces $x = y$. From this observation one can show:

Theorem 5.1 *Let $C \in \mathcal{C}_i$. Suppose that $\theta \in G = \mathrm{Aut}(T)$ stabilizes the equivalence classes of $\sim_{C,i}$ (i.e., $v \sim_{C,i} v^\theta$ for all $v \in L$). Then, for all $j > i$ and $C' \in \mathcal{C}_j$ with $C' \subset C$,*

$$Q(C'\theta, j) \rightarrow Q(C', j).$$

Hence, all descendents of the level i node C' in SB(T) *may be pruned.*

Proof: Let C', j be as indicated. Let $g \in C'$. We must show that the conjunction of the predicates

$$(V^{\phi\theta})_{j-1} = V_{j-1} \quad \rightarrow \quad v_j \leq v_j^{\phi\theta}, \tag{1}$$

$$(V^{\phi})_{j-1} \quad = \quad V_{j-1} \tag{2}$$

imply $v_j \leq v_j^\phi$.

Now, (2) implies $V_i^\phi = V_i$, which implies equality of all the variables in each the equivalence classes of $\sim_{C,i}$. But, since θ stabilizes these classes, this implies $V^\theta = V$ (i.e., the conjunction of all $v^\theta = v$), and, therefore, $V^{\phi\theta} = V^\phi$. Hence, (2) and (3) imply $v_j \leq v_j^\phi$ as required. □

In applying the condition globally, we need to be sure that we do not overprune.

It is safe to prune in all instances where $\sim_{C\theta,i} \neq \sim_{C,i}$, in which case, $\sim_{C\theta,i}$ is a proper refinement of $\sim_{C,i}$ (for C, θ, i as in the theorem, $v \sim_{C\theta,i} w$ always implies $v \sim_{C,i} w$).

In the case that $\sim_{C\theta,i} = \sim_{C,i}$, one could prune provided that $C\theta \prec C$ (there is an induced lexicographic ordering on $\mathcal{C}_i$, which is easily seen via the partial-permutation interpretation of cosets).

Although some technical extensions to this theorem are possible, this formulation is particularly useful because suitable θ can be found using standard tools of computational group theory. The computation employs "set-stabilizer" techniques that are closely related to graph-isomorphism methods. In particular, methods of Luks [1982] are guaranteed to exhibit suitable θ, if such elements exist, in polynomial time under various conditions,

including boundedness of the equivalence classes of $\sim_{C,i}$. As it turns out, in practical computation, this is rarely a difficult problem anyway and is are generally considered to have efficient implementations at least for the cases corresponding to $i \leq 10000$ [6].

One case in which pruning is particularly effective is when the symmetry group of the theory is the full symmetric group (that is, any permutation of L is a symmetry of the theory). In this case the symmetry tree is of size $n!$, but after pruning only n^2 nodes remain. To see why this happens, note that since any perturbation is a symmetry, for any $C \in C_i$ there will always be a $\theta \in \mathrm{Aut}(T)$ that stabilizes the equivalence classes of $\sim_{C,i}$. So for any node C we can always delete *all* it's descendents (so long as we have not already deleted the node $C\theta$!). If one prunes while the tree is being generated, the entire pruned tree (and thus the symmetry-breaking predicate) can be generated in n^2 time. The resultant predicate is not minimal. It turns out that if $V = (v_1, \ldots, v_m)$ then the predicate that one generates consists of a clause $v_i \rightarrow v_j$ for any i, j such that $1 \leq i < j \leq n$. There are obvious polynomial time simplifications that will reduce this to a linear number of clauses, but it is not clear how useful these simplifications will be in the general case.

Remark. Other, less extreme, cases can be constructed in which pruning is effective. However, there are also cases in which the symmetry tree is not prunable to polynomial size. The existence of such cases is a consequence (assuming P$\neq$NP) of theorem 3.2; in fact, the theorem suggests, more strongly, that for some theories, there is no tractable lex-leader predicate since lex-leader *verification* is NP-hard. However, we can also directly construct theories where the symmetry tree cannot be pruned to polynomial size even though the lex-leader problem is in polynomial time (the algorithm uses the "string canonization" procedure of [3], applicable because the group turns out to be abelian). The existence of the polynomial time algorithm, in turn, guarantees we can find *some* symmetry-breaking predicate in polynomial time even though SB(T) is useless for this purpose. Details will appear in a future paper.

6 Approximation

If the symmetry tree is of exponential size, and no pruning is possible, then a natural approach is to generate just a part of the tree, and from this smaller tree generate partial symmetry-breaking predicates. We call a predicate P a *partial symmetry-breaking predicate* for a theory T if the models of P consist of at least one member of each of the symmetry equivalence classes of the truth assignments of the variables in T.

We can thus add P without changing the soundness or completeness of the subsequent search. The trade-off here is that the search engine may visit multiple nodes that are equivalent under some symmetry of T. In essence, then, approximating the symmetry-breaking predicate trades time spent generating symmetry-breaking predicates for time in the search engine.

In the next section we discuss various approaches to generating partial symmetry-breaking predicates.

7 Experimental Results

We have implemented a prototype system that takes a propositional theory in clausal form and constructs an approximate symmetry breaking formula from it. The implementation consists of the following steps:

1. The input theory is converted into a graph such that the automorphisms of the graph are exactly the symmetries of the theory. This is done using the construction given in [8]. There are three "colors" of vertices in this graph, the vertices representing positive literals, those representing negative literals, and those representing clauses. Graph automorphisms are constrained to always map nodes to other nodes of the same color. We also add edges from each literal to each clause that it appears in. These edges (together with the node colorings) guarantee that automorphisms of the graph are symmetries of the theory.[5]

2. We find the generators of the automorphism group of the graph using McKay's graph isomorphism package, nauty [21]. nauty is very fast in practice though there are known examples of infinite classes of graphs which drive nauty to provably exponential behavior [23].

3. From the generators of the automorphism group we construct the symmetry tree and then generate the symmetry-breaking predicate. As expected, in many cases computing the entire symmetry-breaking predicate is computationally infeasible. We use several approximations to compute partial symmetry-breaking predicates:

   - generating predicates for just the generators returned by nauty,
   - building the symmetry tree to some small depth and generating predicates for this smaller tree, and
   - generating random group elements and writing predicates for only those elements.

   An alternative, not yet implemented, would be to use pruning rules such as those from section 5 (though these obviously will not work in all cases).

In the experiments below we generally compare the run time for testing the satisfiability of the input theory alone and conjoined with the symmetry-breaking predicate. In all cases SAT checking was done using the TABLEAU algorithm [9] and run times are "user" time. All code is written in C.

7.1 Experiment 1: The pigeonhole problem

The pigeonhole problem $PHP(n, n-1)$ is the following: place n pigeons in $n-1$ holes such that each pigeon is assigned to a hole and each hole holds at most one pigeon. This problem

[5] For efficiency we special-case binary clauses by representing $x \vee y$ with a link directly from x to y (instead of creating a node for the binary clause and linking x and y to it). This is important because some of the instances we consider have a huge number of binary clauses and some of the algorithms that follow are quadratic, or worse, in the number of nodes.

is obviously unsatisfiable. We study this problem because it is provably exponentially hard for any resolution based method, but is tractable using symmetries. A typical encoding of the problem is to have variables $\{P_{ij} | 1 \leq i \leq n, 1 \leq j \leq (n-1)\}$ where P_{ij} is taken to mean that pigeon i in hole j. $PHP(n, n-1)$ is then:

$$(\forall i \forall j \forall k \ (j \neq k) \Rightarrow (\overline{P_{ij}} \vee \overline{P_{ik}})) \wedge$$
$$(\forall i \vee_{1 \leq j \leq (n-1)} P_{ij}) \wedge$$
$$(\forall j \vee_{1 \leq i \leq n} P_{ij})$$

Since all the pigeons are interchangeable and all the holes are interchangeable, the automorphism group of $PHP(n, n-1)$ is the direct product of 2 symmetric groups. The order of this group $(n!(n-1)!)$ prohibits the full use of the symmetry tree. Furthermore, as we demonstrate in the Appendix (see final remark), pruning as in section 5 cannot help in this case. Hence, for these experiments, we generate only those predicates that are associated with the generators of the automorphism group (or, more specifically, the set of generators returned by nauty). For PHP, such generators are not only determined in polynomial time, but also serve to break all symmetries. (Of course, in general, the predicates associated with generators of Aut(T) do not suffice to break all symmetries.)

The run times for various sizes of the n are shown in figure 3. Run times are on a Sparc 10:51

It is difficult to tell from the run-time data what the scaling is, but it turns out that we can show analytically that every step of our implementation is in polynomial time. The input theory can be represented as a graph with $3n^2 - 2n + 2$ vertices[6]. nauty takes this graph as input and finds the generators of its automorphism group. To do this nauty builds a search tree in which each node is a coloring of the vertices which is a suitable refinement of the coloring of the parent node.[7] The time that nauty spends on each node is polynomial in the size of the input graph. So to show that nauty runs in polynomial time it suffices to show that the number of nodes is polynomial. The proof requires a discussion of the details of the internals of nauty that is beyond the scope of this paper, but one can show that for this problem nauty expands exactly $2n^2 - 3n - 1$ nodes. Computing symmetry-breaking predicates for the generators is obviously in polynomial time. The last step is SAT checking which is, in general, exponential, but for these theories, augmented with the symmetry-breaking predicates, there was no need to run TABLEAU: a proof of unsatisfiability was obtained by a polynomial time simplification procedure that is used as a front-end to TABLEAU.

7.2 Example 2: N-queens

The n-queens problem has been well studied in the CSP literature, but we include it here as a prototypical example of a problem with a small number of geometric symmetries. The

[6]The theory actually has $O(n^3)$ clauses, but many of these clauses are binary clauses that become edges in the graph rather than nodes

[7]A refining of a vertex coloring C is another vertex coloring $\hat{C}$ such that if vertex i and j have the same color in $\hat{C}$ then they have the same color in C.

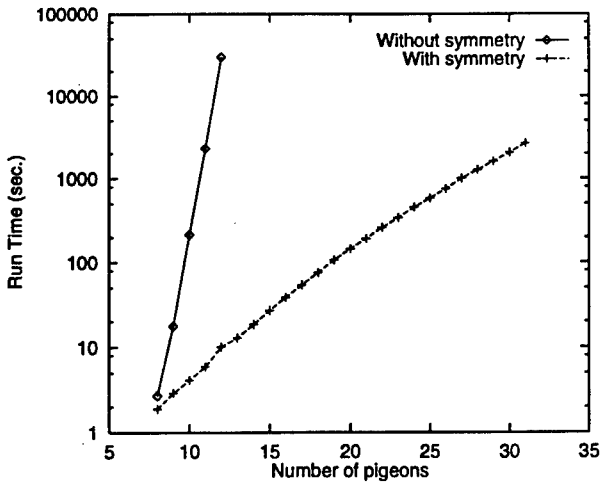


Figure 3: Run times for the pigeon-hole problem with and without symmetry. Note that the y axis is log scaled.

problem is to place n queens on a n by n chess board such that no two queen can attach each other. N-queens has 8 symmetries and for any size board the full symmetry tree has eight leaves.

As one can see from the construction in section 4, nodes at depth i in the symmetry tree generate clauses of length linear in i. It turns out that long clauses are of very little use to a satisfiability checker like TABLEAU, so for these experiments we cut off the symmetry tree at depth 20 and generate predicates only up to this depth. The results are shown in figure 4. Run times are for a Sparc 5.

We know that N-queens is a somewhat delicate problem in that reordering the clauses in the input can drastically change the behavior of SAT checkers (especially as n is increased). Therefore, we took each theory (with and without symmetry-breaking predicates) and randomly perturbed the order of the clauses (and of the variables within the clauses) 50 times.[8][9] We then ran TABLEAU on each permuted theory. Figure 5 shows the average run times. As can be seen, the qualitative nature of the results has not changed but a fair amount of noise has been removed.

8 Related Work

[11] discusses the elimination of interchangeable values in constraint satisfaction problems. [5] discuss an algorithm for backtracking search in the presence of symmetry. In their approach the search engine is modified so that at each node in the search tree a test

[8]Obviously these perturbations have nothing to do with symmetries of the theory. The idea here is only to average out the cases where the SAT checker gets "lucky" and stumbles on a model almost immediately.

[9]For 23 queens the data given is for 20 perturbations.

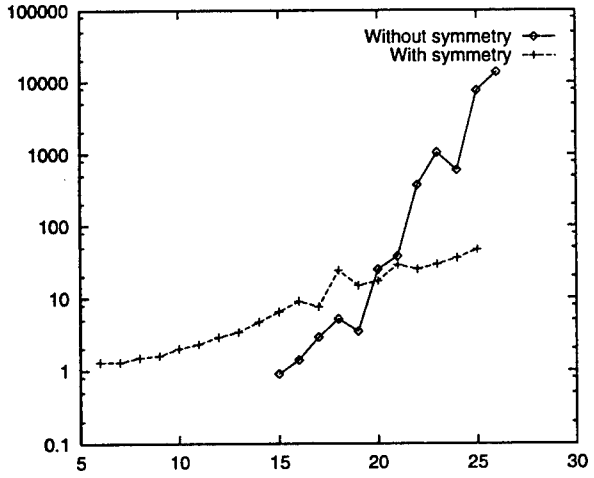


Figure 4: Run times for n-queens with and without symmetry. Note that the y-axis is log scaled.

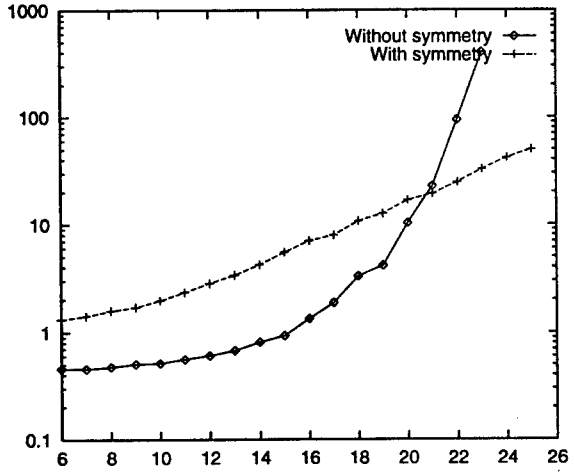


Figure 5: Average run times over 50 random permutations with and without symmetry. Note that the y-axis is log scaled.

is done to determine whether the node is lex-least under the symmetries of the theory. [16] discusses the idea of using symmetries to reduce the lengths of resolution proofs. He uses the "rule of symmetry" which asserts that if θ is symmetry of a theory Th, and one can show that p follows from Th, then $\theta(p)$ also follows from Th (since the proof could be repeated with each step s replaced by $\theta(s)$). Tour and Demri [1995] show that

Krishnamurthy's method is NP-complete in general, but that a restriction of it is equivalent to graph isomorphism. Their restricted method appears to be the same as that discussed by Crawford [8], and Tour and Demri's proof of graph isomorphism uses essentially the same construction used by Crawford. Benhamour and Sais [1992] also discuss techniques for making use of symmetries within specially designed search engines. The most successful uses of symmetry in reducing search spaces is surely in the large and growing literature on the application of automorphism groups to combinatorial problems (see, *e.g.*, [7, 18, 17, 19]). This work has made impressive contributions to the discovery and classification of designs and to the study of combinatorial optimization problems.

9 Conclusion

This work has shown how symmetries can be utilized to add additional constraints, symmetry-breaking predicates, to search problems. These constraints ensure that the search engine never visits two points in the search space that are equivalent under some symmetry of the problem. Complexity results suggest that generating symmetry-breaking predicates will be intractable in the general case. However, partial symmetry breaking can be done in polynomial time (assuming the associated graph isomorphism problem is tractable). Preliminary experiments have been completed showing that partial symmetry breaking is effective on prototypical constraint-satisfaction problems. Work continues on more realistic applications, such as applying these techniques to propositional encodings of planning problems [15].

Appendix

Theorem 3.2 is a direct consequence of the NP-completeness of

Problem. MAXIMUM IN MODEL CLASS (MMC)
Input: A theory T over L; an ordering of L; $M \in \mathcal{M}(T)$.
Question: Does there exist $\theta \in \mathrm{Aut}(T)$ such that ${}^{\theta}M < M$?

We shall demonstrate completeness by a reduction from CLIQUE [12]. We make use of an intermediate problem, which is interesting in its own right. For lexicographic ordering, we consider $m \times n$ matrices, as mn-tuples, taking the rows in succession.

Problem. MAXIMUM INCIDENCE MATRIX. (MIM)
Input: An incidence matrix A for a graph Γ.
Question: Is there an incidence matrix B for Γ such that $B > A$ with respect to lexicographic ordering.

Recall that a $|V| \times |E|$ incidence matrix of $\Gamma = (V, E)$ is determined by specified orderings of V and E, wherein $A_{ij} = 1$ or 0 according to whether or not the ith vertex lies on the jth edge. Two such matrices are then related by permutations of the rows and columns. Thus, in particular, the NP-completeness of MIM establishes the NP-completeness of the problem of the existence of a matrix B, obtained from a given {0,1}-matrix A by permuting rows and columns, such that $B > A$.

Lemma 9.1 *MIM is NP-complete.*

Proof: MIM is in NP since suitable orderings of V and E can be guessed and verified. For the completeness, we reduce CLIQUE to MIM. Suppose we are given an instance (Γ, K) of CLIQUE, wherein $\Gamma = (V, E)$ is a graph and K is a positive integer; the relevant question is whether Γ contains a K-clique (i.e., a complete subgraph on K vertices). We may assume $|V| > K > 3$.

We augment Γ to a graph $\hat{\Gamma} = (\hat{V}, \hat{E})$ as follows. Fix an ordering $v_1, v_2, \ldots, v_{|V|}$ of V. The additional vertices comprise sets W, X, and Y, disjoint from V and from one another. We describe these along with new edges:

1. W is a complete graph on K vertices, $\{w_1, w_2, \ldots, w_K\}$, ordered as indicated by the subscripts.

2. For each $w_i \in W$, join w_i to each vertex in a new set X_i of size $|V|(|V|+1)/2$; $X = \bigcup_i X_i$ is ordered so that X_i precedes X_j for $i < j$. Only X_1 is joined to V and this is done by joining the first $|V|$ elements of X_1 to v_1, the next $|V|-1$ elements to v_2, the next $|V|-2$ to v_3, etc.

3. The set Y, joined only to V, provides a new common neighbor for each pair of vertices V and also ensures the degree of each $v \in V$ is maximal in $\hat{\Gamma}$. Thus, for $1 \leq i < j \leq |V|$, we create a new vertex y_{ij} and add edges joining it to v_i and v_j. Finally, for each $v_i \in V$, join v_i to the vertices in a new set Y_i chosen so as to bring the total degree of v_i to precisely $d = K - 1 + |V|(|V|+1)/2$ (which is also the degree of each $w \in W$). Let $Y = \{ y_{ij} \mid 1 \leq i < j \leq |V| \} \cup \bigcup_i Y_i$. We order Y so that: for $i < j$, y_{ij} precedes Y_i; for $i < i' < j$, Y_i precedes $y_{i'j}$; and for $i < j < j'$, y_{ij} precedes $y_{ij'}$.

The vertices $\hat{V} = W \cup X \cup V \cup Y$ in the resulting graph $\hat{\Gamma}$ are ordered in the sequence $WXVY$ with the orders within each of the four segments as indicated above. By construction, $\hat{\Gamma}$ has a K-clique, W. Observe, however, that $\hat{\Gamma}$ has a *second* K-clique iff Γ had a K-clique, since the vertices in $X \cup Y$ have degree at most 2.

For the instance of MIM, we take A to be the lexicographically greatest incidence matrix of $\hat{\Gamma}$ with respect to the indicated ordering of the vertices. In this regard, note that the maximum incidence matrix for a *given* vertex ordering is obtainable in polynomial time.

We claim that A is the maximum incidence matrix for $\hat{\Gamma}$ if W is first in the ordering of $\hat{V}$. For, suppose A' is the maximum such matrix. Then rows $K+1$ through $K+|X|$ of A' must again correspond to X, the only other vertices directly joined to W; which forces these rows to duplicate their counterparts in A. The next $|V|$ rows in A' must now correspond to V, since these are the only remaining vertices directly joined to Y; suppose these rows correspond to the ordering $v_{i_1}, v_{i_2}, v_{i_3}, \ldots$ of V. Since the v_{i_1} row in A' cannot be exceeded by the v_1 row in A, v_{i_1} must also be joined to $|V|$ elements of X, in fact, to the first $|V|$ elements; so $i_1 = 1$. Similarly, $i_2 = 2$, $i_3 = 3$, etc., so that V remains ordered as before. But, given this ordering of V, the maximality of A' requires Y to be ordered as specified above (except for irrelevant reorderings within each Y_i). Thus, $A' = A$, proving the claim.

We need to show that $\hat{\Gamma}$ has a second K-clique iff $\hat{\Gamma}$ has an incidence matrix $B > A$.

Suppose $\hat{\Gamma}$ has a K-clique $W' \subseteq V$. Reorder $\hat{V}$ so that W' comprises the first K elements and the $(K+1)$st element is the common neighbor in Y of the first two elements of W'. The matrix B induced by the new order agrees with A in the first K rows but its $(K+1)$st row exceeds that of A (the $(K+1)$st row of A begins $0^{K-1}10^{d-2}0$, while the $(K+1)$st row of B begins $0^{K-1}10^{d-2}1$). Hence $B > A$.

Conversely, let B be the maximum incidence matrix for $\hat{\Gamma}$ and suppose that $B > A$. Since the first K rows of A recorded a K-clique consisting entirely of vertices of maximum degree d in $\hat{\Gamma}$, this segment cannot be strictly exceeded. Hence, the first K vertices in the ordering that produces B must form a clique. Since $B > A$, this clique is not W. □

Remark. We trust that the above demonstration will discourage finding-lex-leading-incidence-matrices as an approach to finding canonical forms for graphs and, thereby, to graph isomorphism.

Lemma 9.2 *MMC is NP-complete.*

Proof: MMC is in NP since, one can guess θ, if it exists, and verify both $\theta \in \mathrm{Aut}(T)$ and ${}^{\theta}M < M$ in polynomial time. We reduce MIM to MMC as follows. Suppose the $m \times n$ matrix A constitutes an instance of MIM. Let X and Y be sets of size m, n respectively We describe a theory T on the set $L = X \times Y$ of variables, namely,

$$T = \bigwedge_{\substack{x,x' \in X \\ y \in Y}} ((x,y) \vee (x',y) \vee \overline{(x',y)}) \quad \wedge$$
$$\bigwedge_{\substack{x \in X \\ y,y' \in Y}} (\overline{(x,y)} \vee (x,y') \vee \overline{(x,y')})$$

Trivially, T is a tautology. One verifies that $\mathrm{Aut}(T)$ is $\mathrm{Sym}(X) \times \mathrm{Sym}(Y)$ acting on $X \times Y$ in the natural way. We fix orderings $x_1, x_2, \ldots, x_m$ and $y_1, y_2, \ldots, y_n$ of X and Y respectively, and let $X \times Y$ be ordered lexicographically. The $m \times n$ $\{0,1\}$-matrix A yields a model M of T wherein $M((x_i, y_j)) = 1 - A_{ij}$ (the $0,1$ reversal to accommodate a conversion from maximal matrices to minimal models). There is a natural correspondence between row (respectively, column) permutations and $\mathrm{Sym}(X)$ (respectively, $\mathrm{Sym}(Y)$). Thus, if B is obtained from A via a row permutation and a column permutation, then the permutation pair yield $\theta \in \mathrm{Sym}(X) \times \mathrm{Sym}(Y)$. It follows directly that $B > A$ iff ${}^{\theta}M < M$, establishing the reduction of MIM to MMC. □

Remark. This particular proof of the NP-completeness of MMC was chosen so as to show that the problem remains NP-complete even when L can be identified with some $X \times Y$ and $\mathrm{Aut}(T) = \mathrm{Sym}(X) \times \mathrm{Sym}(Y)$. Note that the pigeonhole problem engenders a theory of exactly this type. (the fact that $|X| = |Y| - 1$ in PHP is not significant – routine padding could be used to force this restriction in MIM). Thus, we have demonstrated the futility, for PHP, of pruning via methods, like that of section 5, which rely solely on information contained in $\mathrm{Aut}(T)$ and $\mathrm{SB}(T)$.

Acknowledgements

Effort sponsored by the Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory, Airforce Materiel Command, USAF, under agreement numbers

F30602-93-C-0031 and F30602-95-1-0023, by AFOSR under agreement numbers F49620-92-J-0384 and F49620-96-1-0335, and by NSF under agreement number IRI-94 12205.

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency (DARPA), the Air Force Research Laboratory, or the U.S. Government.

The work benefited from various discussions over the years with many people including Bart Selman, Steve Minton, Takunari Miyazaki, David Etherington, David Joslin, and all the members of CIRL.

References

[1] L. Babai. Automorphism groups, isomorphism, reconstruction. In L. Lovász R. L. Graham, M. Grötschel, editor, *Handbook of Combinatorics*, chapter 27, pages 1447–1540. North-Holland – Elsevier, 1995, 1995.

[2] L. Babai and L. Kučera. Canonical labelling of graphs in linear average time. In *Proceedings of the Twentieth IEEE Conference on Foundations of Computer Science*, pages 46–49, 1979.

[3] László Babai and Eugene M. Luks. Canonical labeling of graphs. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, pages 171–183, Boston, Massachusetts, 25–27 April 1983.

[4] Belaid Benhamou and Lakhdar Sais. Theoretical study of symmetries in propositional calculus and applications. In D. Kapur, editor, *Automated Deduction: 11th International Conference on Automated Deduction (CADE-11)*, Lecture Notes in Artificial Intelligence, pages 281–294. Springer-Verlag, 1992.

[5] Cynthia A. Brown, Larry Finkelstein, and Paul W. Purdom. Backtrack searching in the presence of symmetry. In T. Mora, editor, *Applied algebra, algebraic algorithms and error correcting codes, 6th international conference*, pages 99–110. Springer-Verlag, 1988.

[6] G. Butler. *Fundamental algorithms for permutation groups.* Lecture notes in computer science. Springer-Verlag, 1991.

[7] G. Butler and C. W. H. Lam. A general backtrack algorithm for the isomorphism problem of combinatorial objects. *Journal of Symbolic Computation*, 1(4):363–382, 1985.

[8] James Crawford. A theoretical analysis of reasoning by symmetry in first-order logic (extended abstract). In *Workshop notes, AAAI-92 workshop on tractable reasoning*, pages 17–22, 1992.

[9] James Crawford and Larry Auton. Experimental results on the crossover point in random 3sat. *Artificial Intelligence*, 81, 1996.

[10] Thierry Boy de la Tour and Stéphane Demri. On the complexity of extending ground resolution with symmetry rules. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, volume 1, pages 289–295, 1995.

[11] Eugene G. Freuder. Eliminating interchangeable values in constraint satisfaction problems. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 227–233, 1991.

[12] M.R. Garey and D.S. Johnson. *Computers and Intractability.* W.H. Freeman and Co., New York, 1979.

[13] John Gaschnig. Performance measurement and analysis of certain search algorithms. Technical Report CMU-CS-79-124, Carnegie-Mellon University, 1979.

[14] Matthew L. Ginsberg. Dynamic backtracking. *Journal of Artificial Intelligence Research*, 1:25–46, 1993.

[15] David Joslin and Amitabha Roy. Exploiting symmetry in plan generation. Unpublished manuscript, 1996.

[16] B. Krishnamurthy. Short proofs for tricky formulas. *Acta Informatica*, 22:253–275, 1985.

[17] C. W. H. Lam. Applications of group theory to combinatorial searches. In L. Finkelstein and W. M. Kantor, editors, *Groups and Computation, Workshop on Groups and Computation*, volume 11 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 133–138, 1993.

[18] C. W. H. Lam and L. Thiel. Backtrack search with isomorph rejection and consistency check. *Journal of Symbolic Computation*, 7(5):473–486, 1989.

[19] R. Laue. Construction of groups and the constructive approach to group actions. In S. Walcerz T. Lulek, W. Florek, editor, *Symmetry and Structural Properties of Condensed Matter in Proceedings of the Third International School on Theoretical Physics*, pages 404–416. World Scientific - Singapore, New Jersey, London, Hong Kong, 1995.

[20] E. M. Luks. Isomorphism of graphs of bounded valence can be tested in polynomial time. *J. Comp. Sys. Sci.*, 25:42–65, 1982.

[21] B. D. McKay. *Nauty user*'s guide, version 1.5. Technical Report TR-CS-90-02, Department of Computer Science, Australian National University, Canberra, 1990.

[22] Steven Minton, Mark D. Johnston, Andrew B. Philips, and Philip Laird. Solving large-scale constraint satisfaction and scheduling problems using a heuristic repair method. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 17–24, 1990.

[23] Takunari Miyazaki. The complexity of McKay's canonical labeling algorithm. In L. Finkelstein and W. M. Kantor, editors, *Groups and Computation II, Workshop on Groups and Computation*, volume to appear of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, 1996.

[24] Bart Selman, Hector Levesque, and David Mitchell. A new method for solving hard satisfiability problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 440–446, 1992.

Appendix F

Exploiting Symmetry in Lifted CSPs*

David Joslin
Computational Intelligence Research Laboratory
University of Oregon
Eugene, OR 97403
joslin@cirl.uoregon.edu

Amitabha Roy
Dept. of Computer and Information Science
University of Oregon
Eugene, OR 97403
aroy@cs.uoregon.edu

February 4, 1999

Abstract

When search problems have large-scale symmetric structure, detecting and exploiting that structure can greatly reduce the size of the search space. Previous work has shown how to find and exploit symmetries in propositional encodings of constraint satisfaction problems (CSPs). Here we consider problems that have more compact "lifted" (quantified) descriptions from which propositional encodings can be generated. We describe an algorithm for finding symmetries in lifted representations of CSPs, and show sufficient conditions under which these symmetries can be mapped to symmetries in the propositional encoding. Using two domains (pigeonhole problems, and a CSP encoding of planning problems), we demonstrate experimentally that the approach of finding symmetries in lifted problem representations is a significant improvement over previous approaches that find symmetries in propositional encodings.

1 Introduction

As others have shown [Benhamou & Sais, 1992; Benhamou, 1994; Crawford *et al.*, 1996; Brown, Finkelstein, & Purdom, 1988], when search problems have large-scale symmetric structure, detecting and exploiting that structure can greatly reduce the size of the search space. Symmetries can arise in a variety of ways. In planning problems, for example, interchangeable resources are one source. In generating a plan for flying a package from one city to another, it makes no sense to waste time trying to decide between two interchangeable

*This paper appeared in *Proc. AAAI-97.*

airplanes. If the planes do not differ in any significant way, the planner should just pick one. If it turns out that the goal cannot be achieved with the selected plane, there is no need to backtrack and try to generate a plan with the other airplane. The symmetry between the planes guarantees that any solution to the planning problem has a symmetric solution found by simply exchanging the two planes everywhere they are mentioned in the plan. Failing to recognize such symmetries can make the search very inefficient.

Previous work has shown how to find and exploit symmetries in propositional encodings of constraint satisfaction problems (CSPs) [Benhamou & Sais, 1992; Benhamou, 1994; Crawford *et al.*, 1996]. Here we consider problems that have "lifted" descriptions. Although most constraint solvers require propositional encodings, it is often natural to describe problems using axioms quantified over the objects in the domain. We might define a map coloring problem, for example, by identifying the countries (A, B and C), the available colors (*red* and *blue*), the variables (for each country c, $colorof(c) \in \{red, blue\}$), and the adjacencies (*adj(A,B)*, *adj(B,C)*); all we then need to add are the following axioms:

$$\forall x \forall y \ \ (adj(x,y) \rightarrow adj(y,x))$$

$$\forall x \forall y \ \ (adj(x,y) \rightarrow colorof(x) \neq colorof(y))$$

It is simple to expand these axioms over the finite domains of objects to generate a purely propositional theory. The advantage of working with lifted representations is that they are typically very compact, compared to the corresponding propositional theories that result from expanding the quantified axioms.

We present an algorithm for finding symmetries in lifted problem descriptions, and demonstrate experimentally, in two domains, that this approach is a significant improvement over finding symmetries in the expanded propositional theory, which in turn is a significant improvement over backtracking search that makes no use of symmetry. One of the domains examined is the pigeonhole problem, for which we compare our results to those of [Crawford *et al.*, 1996]. We also examine a logistics planning domain, using a CSP encoding of planning problems from [Kautz & Selman, 1996]. The same approach would be directly applicable to other planners that represent planning problems (or parts of them) as CSPs [Kautz & Selman, 1996; Joslin, 1996; Joslin & Pollack, 1996; Yang, 1992].

2 Detecting and exploiting symmetries

Planning problems, represented as CSPs, are good examples of problems that have natural, lifted descriptions. A number of planners have applied constraint reasoning to some degree, but some recent approaches have focused on transforming entire planning problems into CSPs [Joslin, 1996; Joslin & Pollack, 1996; Kautz & Selman, 1996; Ginsberg, 1996].

A planning problem can be described by the initial and goal states, and the axioms that define the legal states and legal state transitions in a domain. For example, in a state-based encoding for a logistics domain, we might have airplanes *a1* and *a2*, and packages *p1* and *p2*, and have the boolean variable *in(p1,a1,3)* be true if and only if package *p1* is in plane *a1* at time $t = 3$. We would similarly have variables *in(x,y,t)* for all packages x, airplanes

y, and time points t. Axioms would describe the legal states and state transitions. One axiom, for example, would be

$$\forall p \forall x \forall y \forall t \; ((at(p,x,t) \wedge at(p,y,t)) \rightarrow x = y)$$

i.e., a plane cannot be in two cities at the same time. Another axiom might require that if a package is moved from one location to another, it must have been on a plane that made the indicated flight. By grounding out the axioms over the finite domains, and adding the literals that define the initial and goal states, we can produce a propositional theory.

To generalize this construction, we define a *lifted constraint satisfaction problem* to be a tuple $\{D, S, L\}$ where D is a finite, colored set of atoms representing the necessary domains, with a distinct "color" assigned to each type of element, S is a set of ground literals, and L is a restricted first-order theory in which quantification is allowed only over the finite domains. (L is thus a compact representation of a propositional theory.)

In a planning problem, D identifies the "objects" relevant to the problem (airplanes, packages, and time points), S defines the initial and goal states (*at(a1,boston,0)* if plane *a1* is in Boston in the initial state), and L contains the axioms that describe the legal states and legal state transitions in the domain. For the map coloring example, D would define the sets of countries and colors, S would give the adjacency relations, and L would contain the axioms mentioned previously. We call S the *problem description* because, in general, the quantified axioms in L hold across all of the problems in a domain (an airplane can only be in one place at a time), while S describes a specific problem (airplane *a1* is in Boston at time $t = 0$).

When L is grounded over the atoms in D, yielding L_D, and combined with S, we get a propositional encoding of a theory, $\mathcal{T}$. Abusing the notation, we will write $\mathcal{T} = S \wedge L_D$.

Prior approaches such as [Crawford *et al.*, 1996] find and break symmetries in the propositional theory, $\mathcal{T}$. Because $\mathcal{T}$ can be very large, finding symmetries in $\mathcal{T}$ can be computationally intensive. Our algorithm instead finds symmetries in S, which is much smaller than $\mathcal{T}$, and then uses D to map those symmetries to symmetries in $\mathcal{T}$. We show sufficient conditions for this mapping to be valid.

Symmetry detection in a boolean CSP is polynomial time equivalent to graph isomorphism [Crawford, 1992]. Though graph isomorphism is a difficult problem, it is not known to be NP complete nor to be in P. Despite the computational difficulty of the problem, there are group theoretic algorithms, such as NAUTY [McKay, 1990], that work well in practice. Both the algorithm used in [Crawford *et al.*, 1996] and our own algorithm find symmetries by generating a graph from $\mathcal{T}$ or S, respectively, and then invoking NAUTY. The symmetries are then used to generate symmetry-breaking constraints, which are added to the CSP.

We first introduce some notation from group theory. Let G be a group. We write $H \leq G$ to indicate that H is a subgroup of G. We denote by $P(\Omega)$ the symmetric group on the finite set Ω i.e. the set of all $|\Omega|!$ permutations of Ω. Refer to [Wielandt, 1964] for definitions and elementary results on permutation groups.

Let $\mathcal{T}$ be a propositional theory. Let V be an ordered set of variables of $\mathcal{T}$. A permutation $\sigma \in P(V)$ is called a symmetry of $\mathcal{T}$ if it produces the same theory after it permutes the clauses and literals of $\mathcal{T}$. A symmetry σ is said to preserve $\mathcal{T}$. $Sym(\mathcal{T}) = \{\sigma | \sigma \in P(V)$

is a symmetry of $\mathcal{T}\} \leq P(V)$ is called the symmetry group of the theory $\mathcal{T}$. For results on symmetries of propositional theories, see [Crawford *et al.*, 1996].

Consider a problem, $\mathcal{P} = \{D, S, L\}$, where the full propositional theory is $\mathcal{T} = S \wedge L_D$. Let V be the variables of $\mathcal{T}$ in some order. Note also that V consists of predicates over D: for example, in the logistics domain, a variable in $\mathcal{T}$ might be $at(a1, c1, 4)$, true iff plane $a1$ is at location $c1$ at $t = 4$.

A permutation of atoms in the domain D induces a permutation of variables in $\mathcal{T}$. For example, the permutation $(a1, a2)$ maps plane $a1$ to $a2$ and plane $a2$ to $a1$ in D, and will map the variable $at(a1, c1, 0)$ to $at(a2, c1, 0)$ (and vice versa) in V. It will permute all variables in V which mention planes $a1$ or $a2$ in this manner. It will map variables that don't mention planes $a1$ or $a2$ to themselves. This gives us a permutation in $P(V)$. This procedure gives us a mapping from $P(D)$ to $P(V)$ – we say that we can "extend" a permutation of D to a permutation of V.

We first compute a group of permutations, $H \leq P(D)$, of D whose extension $K \leq P(V)$ preserves S i.e. $K \leq Sym(S)$. The following theorem gives a sufficient condition for K to preserve L_D.

Theorem 1 *If L does not mention any domain elements in D, then* any *permutation of D when extended to a permutation of V will preserve L_D.*

The proof is an easy induction on the quantifier depth of L. An universal quantifier over a "colored set" in the domain (e.g airplanes) mentions the airplanes symmetrically in the conjunction that we obtain on grounding out L. An existential quantifier similarly mentions those elements symmetrically in the disjunction. So in particular, if we have found a set of permutations of D which preserve S when extended, they also preserve L_D when extended, since any permutation of D preserves L_D. Therefore, they also preserve $\mathcal{T} = S \wedge L_D$ when extended, because if $T = T_1 \wedge T_2$ where T, T_1 and T_2 are propositional theories, then $Sym(T_1) \cap Sym(T_2) \leq Sym(T)$. Thus by looking at the action of $P(D)$ on the much smaller propositional theory S, we can find a group of symmetries of $\mathcal{T}$.

We now present an algorithm that, given a problem $\mathcal{P} = \{D, S, L\}$, computes symmetries of $\mathcal{T}$ by computing the subgroup $H \leq P(D)$ which preserve S, and extending H to a subgroup $K \leq P(V)$. We assume that L does not mention any elements in D. The algorithm constructs a colored graph that captures the symmetries of S. NAUTY then finds the automorphisms of the graph. Those automorphisms are restricted to the domain elements, D, which we then extend to permutations of V.

1. Let A be a graph, initially empty. The nodes of A will be colored.

2. For each element $x \in D$, add a vertex V_x to A. Two such vertices, V_x and V_y share the same color iff they are objects of the same color (same type) in D. For example, if there are three airplanes $a1$, $a2$ and $a3$ in the domain, we have three vertices with the same color in the graph.

3. For each ground literal in S, add a vertex with edges to the domain elements that the literal mentions. For example, if $at(a1, c1, 0)$ is a literal in S, we add a new

vertex connected to the vertices representing $a1$, $c1$, and time point 0. A vertex for a literal never has the same color as a vertex for a domain element. Two literal vertices share the same color iff they have the same predicate. For example, $at(a1, c1, 0)$ and $at(a3, p5, 0)$ would get the same color.

4. Give the graph thus generated to NAUTY, which returns the generators for the graph's automorphism group. We restrict the generators to the vertices representing the domain elements, so the result will be in $P(D)$. Extend each generator to a permutation in V. Then $K \leq P(V)$ is generated by the extended generators. Since L does not mention any elements of D, the above theorem implies that $K \leq Sym(L_D)$.

In the discussion above, we have imposed the restriction that L not mention any elements of D. This is a sufficient condition for symmetries in the propositional part, S, extending to symmetries in the ground theory $\mathcal{T}$). In the pigeonhole problems used in the experiments discussed below, this condition is met. In the planning problems, however, time points are mentioned by the presence of a "successor" function, necessary to express that a change at one time point affects the world state at the successor time point. The successor function may be a "built-in" function, but it must be treated as if it were explicitly defined for the purpose of determining which domain elements are mentioned in L.

Let $D' \subseteq D$ be the set of elements in D that are not mentioned in L. We do not need to require that L mention no atoms, but only that it mention no atoms in D. Therefore, if we replace D with D' in the algorithm given above, we meet the sufficient condition that L mention no domain elements in the specified set. Any symmetries we find over D' will extend to symmetries in the fully-ground theory. We currently do not automatically restrict the set of elements in D to exclude those that are mentioned in L, but this could easily be automated.

In the worst case, every element in D is mentioned in L, and our approach fails to find any symmetries. That does not mean that no symmetries exist, and it may even be the case that elements are mentioned in L in a way that preserves symmetry. An alternative approach would be to find permutations of D that extend to symmetries in S, disregarding the mention of any elements of D in L. We can then check whether each permutation preserves L_D; this is equivalent to checking whether a given permutation of vertices of a graph is an automorphism of the graph, and can be computed in time quadratic in the size of L_D. Permutations that are not symmetries would then be discarded.

3 Example

Suppose we have three cities (*c1*, *c2*, and *c3*), two packages (*p1* and *p2*), and one airplane (*a*). The plane is initially at *c3*, and the packages are at *c1* and *c2*, respectively. The goal is to transfer package *p1* to city *c2*, and package *p2* to city *c1*. Symmetries should tell us, for example, that if there is no solution that starts by having the plane fly first to city *c1* to deliver package *p1*, then there cannot be a solution that instead starts with the plane flying first to city *c2* to deliver package *p2*.

Assuming that the bound on the length of the plan is $t = 3$, the initial and goal states might be described as follows:

$at(p1, c1, 0)$ $\quad$ $at(p1, c2, 3)$
$at(p2, c2, 0)$ $\quad$ $at(p2, c1, 3)$
$at(a, c3, 0)$

In addition, we have axioms that define the domain, as discussed previously. The axioms mention time points, so the domains we are interested in are restricted to planes, packages and cities.

The algorithm for finding symmetries in the problem begins by building a colored graph with vertices for each of the domain elements: *a*, *c1*, *c2*, *c3*, *p1*, and *p2*. Elements of the same type share the same color. We also have vertices for each of the ground literals that define the initial and goal states, with edges for each domain element mentioned. For example, the node for *at(p1,c1,0)* has edges to the nodes for *p1* and *c1*.

Finding symmetries in the planning problem has now been transformed into the problem of finding automorphisms in the resulting colored graph. In this case, exchanging *c1* with *c2*, and *p1* with *p2*, and the corresponding ground literals as well, gives us a graph that is isomorphic to the original. Therefore, we know that we have a symmetry in the original problem that occurs when we exchange packages *and* exchange cities. Obviously this approach will also detect simpler symmetries, such those that would occur if we had some number of interchangeable airplanes.

To exploit this symmetry, we project it onto the ground version of the theory. For example, exchanging domain element *c1* with *c2* in the lifted theory corresponds to exchanging *at(a,c1,1)* and *at(a,c2,1)* in the ground theory. By making all of the appropriate exchanges in this fashion, we identify a symmetry in the ground theory.

We can then generate symmetry-breaking constraints, as described in [Crawford *et al.*, 1996]. For example, since we have a symmetry that exchanges *at(a,c1,1)* and *at(a,c2,1)*, one of the symmetry-breaking constraints might be:

$$at(a, c1, 1) \rightarrow at(a, c2, 1))$$

This would be added to the theory, with the effect of requiring that if we pick just one of these alternatives, it must not be *at(a,c1,1)*. In other words, if the solver proves that no solution has *at(a,c2,1)* (i.e., we can't solve the problem by going to city *c2* first), then the symmetric alternative of going to city *c1* first is eliminated also, since the new clause prevents us from having *at(a,c1,1)* without also having *at(a,c2,1)*. By adding the new constraints, symmetric choices are forced to be made in one particular way, thus allowing a systematic solver to avoid some redundant effort in the search.

4 Experimental results

The first domain we examine is the pigeonhole problem, in which N pigeons are to be placed in $N - 1$ holes [Benhamou & Sais, 1992]. The lifted description consists of the domains for pigeons and holes, an axiom requiring that only one pigeon can be assigned to a given

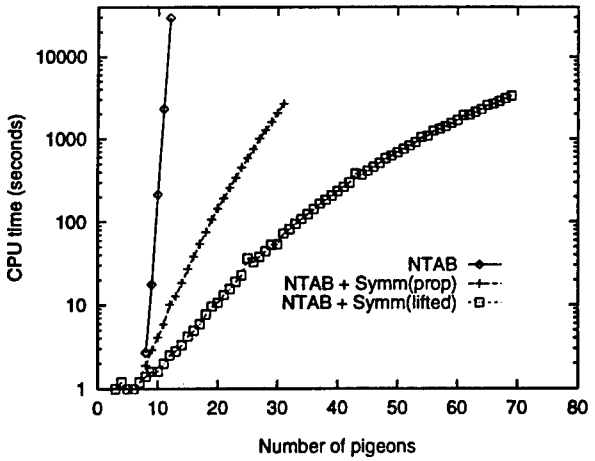


Figure 1: Pigeonhole problems (y-axis is log scaled)

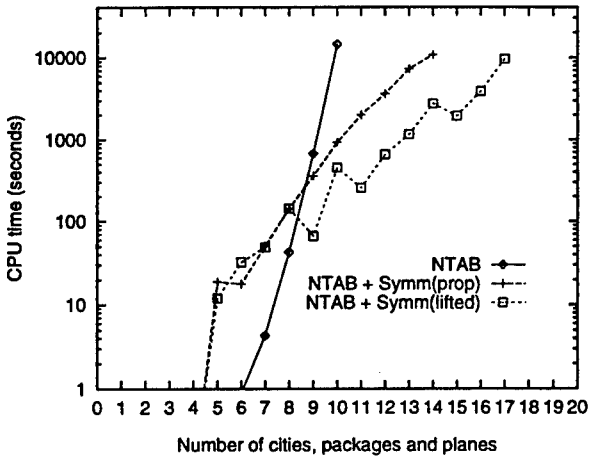


Figure 2: Logistics problems (y-axis is log scaled)

hole, and an axiom requiring that every pigeon be assigned to some hole. The boolean variables are *in(p,h)* for all pigeons, p, and all holes, h. Results were presented in [Crawford *et al.*, 1996] comparing NTAB [Crawford, 1996] (a solver based on Tableau) without using symmetry, to NTAB using symmetry-breaking constraints derived from symmetries found by NAUTY in the propositional theory. In figure 1 we show data for these two approaches, as well as our own results using symmetries discovered in the lifted description. The CPU time for each data point is the time required to prove that the theory is unsatisfiable. (All CPU times reported here are from experiments run on a SPARCstation 10.) As the graph shows, the lifted approach is significantly faster than finding symmetries in the propositional theory, and both easily outperform NTAB without symmetry breaking.

Our second set of experiments uses logistics planning problems based on one of the domains used in [Kautz & Selman, 1996]. Although Kautz and Selman solve only ground theories, they essentially use a lifted representation to generate those ground theories. For

our experiments, we used a reconstruction of that lifted representation [Bedrax-Weiss, 1996].

Each problem in our test set has N packages, each in a different city, and $N-1$ planes. Each package must be delivered to a city other than its starting point. The planes all start from a common location that is neither a starting point nor a destination for any of the packages. Because there are not enough airplanes for all of the packages, a valid plan will require at least one plane to make two trips. Problems were generated from $N = 4$ up to $N = 17$.

A polynomial-time preprocessing step compacts the problem by looking for "failed literals," where a failed literal is defined to be one that, when added to the theory, leads to a contradiction by unit resolution. NTAB uses dynamic backtracking [Ginsberg, 1993] without variable re-ordering. In combination, these techniques in the current version of NTAB improve considerably upon the version used by Kautz and Selman in [Kautz & Selman, 1996]. The logistics problems they describe that were unsolvable by the earlier version of NTAB with a search limit of ten hours are now solved within a few minutes (without using symmetries).

Non-systematic solvers such as WSAT have so far proven to be much more effective at solving satisfiable planning problems than systematic search techniques, with or without symmetry breaking. Non-systematic solvers are of no use, however, in proving that a problem is unsatisfiable. To find an optimal (minimum length) solution, one must find a solution of length t, *and* prove that no solution exists for length $t-1$. For this reason, we anticipate that systematic and non-systematic solvers would be used in parallel, and here we focus on unsatisfiable cases.

Figure 2 shows experimental results with a bound on plan length that allows the planes to make only one trip, making the problems unsatisfiable. The times shown do not include the preprocessing time, which was common to all three of the techniques employed here. Times shown for NTAB with symmetry breaking include the time required to generate the graph, run NAUTY, generate the symmetry breaking constraints, run another compression step on the augmented problem, and finally to run NTAB until the problem is shown to be unsatisfiable. In the propositional case, the graph is generated on the full, expanded theory. In the lifted case, the graph is generated on the propositional part of the lifted theory, as described previously, then the symmetries are mapped to the propositional theory, for which symmetry breaking constraints are generated. As the results show, breaking the symmetries can significantly improve upon the time required to prove that a problem is unsatisfiable.

Our approach of finding symmetries in the compact problem description allows us to find them very quickly; on the largest logistics problem, only about 30 seconds (out of 9650 seconds total) were spent finding symmetries.

5 Related work

The approach described here finds symmetries in the problem description, and the approach described in [Crawford *et al.*, 1996] finds symmetries in the full propositional expansion of the problem, but both break those symmetries by adding new constraints. The augmented problem can then be solved by any standard CSP engine. An alternate approach is to

exploit symmetries dynamically, during the search for a solution [Benhamou, 1994; Brown, Finkelstein, & Purdom, 1988; Lam & Thiel, 1989; Lam, 1993]. In such approaches, the reasoning done about symmetries is very tightly coupled to the search techniques themselves. These approaches would likely benefit from our technique of finding symmetries quickly from the problem description.

One potential advantage to our approach is portability; if a new CSP search engine becomes available, the techniques presented here for detecting and breaking symmetries could be applied immediately, because they only depend on being able to augment the CSP with new constraints. We anticipate, however, that using the symmetry information more directly in the search engine will be far more efficient than our current approach of generating symmetry-breaking constraints. The symmetries discovered by our algorithm can be described very compactly, but on all but the smallest of problems it would be infeasible to generate the constraints needed to break all of those symmetries. Using the compact description of the symmetries directly is a much more promising approach.

Little has been done about automatic recognition of symmetries in planning problems, so both the lifted and non-lifted versions are improvements over previous work. Symmetries that arise from interchangeable resources have typically been handled through knowledge engineering. Some planners, for example, allow interchangeable resources to be assigned to resource pools; the planning algorithm implicitly recognizes that resources within a pool are interchangeable [Tate, Drabble, & Dalton, 1994]. Another option has been to provide domain-dependent search control information, causing the planner to select and commit to resource allocations in a way that makes sense for the particular domain. This paper shows that it is possible to find symmetries in planning problems automatically, including (but not limited to) those that arise because of interchangeable resources.

6 Summary and future work

We have shown that it is possible to find and exploit symmetries, such as those that arise from interchangeable resources in planning problems, by analyzing a lifted problem description. Finding these symmetries and adding constraints that break them can be highly effective in reducing the search space.

We hope to look at the possibility of using *approximate* symmetries. Strictly speaking, if two resources differ in any of their properties, they are not symmetric. Not all properties are equally important in all problems, however. For example, two planes that differ slightly in the weight of cargo they can carry may be interchangeable in most problems. Approximate symmetries have also been examined in [Ellman, 1993]. There, for function-finding problems, an approximate symmetry for goal function G must be an exact symmetry for some other goal function, $\hat{G}$ that is a "good approximation" of G, where "good approximation" is taken to mean either that G entails $\hat{G}$, or vice versa.

Although we have focused on planning and simple constraint satisfaction problems here, the approach we have taken can be easily applied to other types of problems as well. In manufacturing scheduling problems, for example, identical items in a production run and multiple production lines can introduce symmetries. Identifying approximate symmetries

could be very helpful here as well; in a production run of cars on an assembly line, for example, there may be few exact duplicates, but some sets of cars may differ only in relatively minor ways. In searching for near-optimal schedules, when it is impossible to exhaustively examine the entire search space, identifying approximate symmetries could be very helpful in focusing the search on alternatives that are not just minor variations on solutions found so far.

As noted, on satisfiable problems local search typically outperforms systematic search, even when the systematic search is assisted by symmetry breaking. An interesting open question is whether symmetry-breaking constraints would be helpful or harmful (or neither) with local search techniques on satisfiable problems. On the one hand, these new clauses eliminate some solutions, so the solution density decreases. On the other hand, the new clauses may discourage local changes that merely move from one non-solution to a symmetric equivalent. Some preliminary experiments suggest that breaking symmetries hurts local search performance, but more work remains to be done.

We also hope to extend the approach taken here to find symmetries in the non-ground portion of lifted CSPs as well. The approach would be to convert the theory to a canonical form, then build a graph such that symmetries in the lifted theory would correspond to symmetries in the graph. There is no *a priori* reason to believe that this would not work, but the issues are subtle.

We hope to pursue these and other directions of research, and that the work presented here will provide a basis for efficient techniques for detecting and exploiting symmetry in problems that are naturally represented as lifted CSPs.

Acknowledgments: This research has been supported by the Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory, Airforce Materiel Command, USAF, under agreement numbered F30602-95-1-0023, the AFOSR (F49620-96-1-0335), and an NSF CISE Postdoctoral Research award (CDA-9625755).

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency (DARPA), the Air Force Research Laboratory, or the U.S. Government.

References

[Bedrax-Weiss, 1996] Bedrax-Weiss, T. 1996. personal communication.

[Benhamou & Sais, 1992] Benhamou, B., and Sais, L. 1992. Theoretical study of symmetries in propositional calculus and applications. In Kapur, D., ed., *Automated Deduction: 11th International Conference on Automated Deduction (CADE-11)*, Lecture Notes in Artificial Intelligence, 281–294. Springer-Verlag.

[Benhamou, 1994] Benhamou, B. 1994. Study of symmetry in constraint satisfaction problems. In *Principles and Practice of Constraint Programming (PPCP-94)*.

[Brown, Finkelstein, & Purdom, 1988] Brown, C. A.; Finkelstein, L.; and Purdom, Jr., P. W. 1988. Backtrack searching in the presence of symmetry. In Mora, T., ed., *Applied algebra, algebraic algorithms and error correcting codes, 6th international conference*, 99–110. Springer-Verlag.

[Crawford *et al.*, 1996] Crawford, J.; Ginsberg, M.; Luks, E.; and Roy, A. 1996. Symmetry breaking predicates for search problems. In *Proc. KR-96.*

[Crawford, 1992] Crawford, J. 1992. A theoretical analysis of reasoning by symmetry in first-order logic (extended abstract). In *Workshop notes, AAAI-92 workshop on tractable reasoning*, 17–22.

[Crawford, 1996] Crawford, J. 1996. Satisfiability techniques for planning problems. Unpublished.

[Ellman, 1993] Ellman, T. 1993. Abstraction via approximate symmetry. In *Proc. IJCAI-93.*

[Ginsberg, 1993] Ginsberg, M. 1993. Dynamic backtracking. *Journal of Artificial Intelligence Research* 1:25–46.

[Ginsberg, 1996] Ginsberg, M. L. 1996. A new algorithm for generative planning. In *Proc. KR-96.*

[Joslin & Pollack, 1996] Joslin, D., and Pollack, M. E. 1996. Is "early commitment" in plan generation ever a good idea? In *Proc. AAAI-96.*

[Joslin, 1996] Joslin, D. 1996. *Passive and active decision postponement in plan generation.* Ph.D. Dissertation, University of Pittsburgh, Pittsburgh, PA.

[Kautz & Selman, 1996] Kautz, H., and Selman, B. 1996. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proc. AAAI-96.*

[Lam & Thiel, 1989] Lam, C. W. H., and Thiel, L. 1989. Backtrack search with isomorph rejection and consistency check. *Journal of Symbolic Computation* 7(5):473–486.

[Lam, 1993] Lam, C. W. H. 1993. Applications of group theory to combinatorial searches. In Finkelstein, L., and Kantor, W. M., eds., *Groups and Computation, Workshop on Groups and Computation*, volume 11 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 133–138.

[McKay, 1990] McKay, B. D. 1990. *Nauty* user's guide, version 1.5. Technical Report TR-CS-90-02, Department of Computer Science, Australian National University, Canberra.

[Tate, Drabble, & Dalton, 1994] Tate, A.; Drabble, B.; and Dalton, J. 1994. Reasoning with constraints within O-Plan2. Technical Report ARPA-RL/O-Plan2/TP/6 Version 1, Artificial Intelligence Applications Institute, University of Edinburgh.

[Wielandt, 1964] Wielandt, H. 1964. *Finite Permutation Groups.* New York: Academic Press.

[Yang, 1992] Yang, Q. 1992. A theory of conflict resolution in planning. *Artificial Intelligence* 58:361–392.

Appendix G

Clustering at the Phase Transition*

Andrew J. Parkes
CIS Dept. and CIRL
1269 University of Oregon
Eugene, OR 97403-1269
parkes@cirl.uoregon.edu

February 4, 1999

Abstract

Many problem ensembles exhibit a phase transition that is associated with a large peak in the average cost of solving the problem instances. However, this peak is not necessarily due to a lack of solutions: indeed the average number of solutions is typically exponentially large. Here, we study this situation within the context of the satisfiability transition in Random 3SAT. We find that a significant subclass of instances emerges as we cross the phase transition. These instances are characterized by having about 85–95% of their variables occurring in unary prime implicates (UPIs), with their remaining variables being subject to few constraints. In such instances the models are not randomly distributed but all lie in a cluster that is exponentially large, but still admits a simple description. Studying the effect of UPIs on the local search algorithm WSAT shows that these "single-cluster" instances are harder to solve, and we relate their appearance at the phase transition to the peak in search cost.

1 Introduction

A phase transition in a physical many-body system is the abrupt change of its macroscopic properties at certain values of the defining parameters. The most familiar example is the water/ice transition in which the fluidity changes abruptly at particular temperatures and pressures. Phase transitions also occur in systems associated with computer science: work in this area started with the remarkable observation [Erdös & Rényi, 1960] that thresholds in properties such as connectivity emerge in large random graphs. Recently, phase transitions have been studied in constraint satisfaction, e. g. see [Cheeseman, Kanefsky, & Taylor, 1991; Mitchell, Selman, & Levesque, 1992; Williams & Hogg, 1993; Kirkpatrick & Selman, 1994; Smith, 1994]. In all these cases we have "control" parameters defining the system, a method to generate an ensemble of problem instances given values of such parameters, and some property A whose existence or not we wish to determine for each problem instance. When

*This paper appeared in *Proc. AAAI-97.*

the problems are sufficiently large, then small variations in the control parameters can mean that the ensembles quickly change from having almost all instances satisfying the property A, to having almost no instances satisfying A.[1]

It can also be that we need to search in order to determine whether or not property A is satisfied (e. g. determining satisfiability for SAT problems) and then the phase transition is typically associated with a peak in the cost of such a search. It becomes easier to find either a witness to A, or a proof of its non-existence, as we move away from the transition. This peak has (at least) two impacts on artificial intelligence. Firstly, even in real problems we might well see phase transitions along with the peak in search cost [Huberman & Hogg, 1987]. Secondly, the development of search algorithms has been plagued by the lack of hard test problems. If we use real problems then the test set is likely to be small and we have the risk of over-fitting. To make the algorithms more robust it is useful to be able to generate a large number of test instances. However, initial attempts at artificial generation gave unrealistically easy instances. This situation was relieved by the discovery that phase transitions can be used as sources of hard artificial problems [Mitchell, Selman, & Levesque, 1992]. This suggests that deeper study of such transitions is relevant to AI.

If we refer to a witness to the property A as a model, then in many random systems it is possible to determine the average number of models in terms of the control parameters. At the phase transition the average number of models is typically *exponential* in the problem size, and this immediately raises the questions:

- how are the many models distributed amongst instances of the ensemble?
- for a particular instance how are the models (if any) distributed in the space of assignments?

Furthermore, even if we restrict ourselves to satisfiable instances and use local search, then the search cost still seems to peak at the phase transition [Hogg & Williams, 1994; Clark *et al.*, 1996]. This is somewhat counter-intuitive because the average number of models per instance does not peak (or even seem to be special in any way), yet we might well expect that the search cost is related to numbers of models.

In this paper we study such issues in the context of the well-studied satisfiability transition in Random 3SAT [Kirkpatrick & Selman, 1994; Crawford & Auton, 1996; Schrag & Crawford, 1996]. Our aim is to provide a finer description of the transition than the coarse description as a boundary between satisfiable and unsatisfiable phases. However, the exponential number of models also makes it difficult to study them directly. Instead we work indirectly by looking at the implicates of the theory and in particular at the distributions associated with the unary prime implicates[2] (UPIs). Our main result is that there is indeed a finer structure observable in the UPI–distributions. As we cross into the unsatisfiable phase then there emerges a large distinct subclass of instances. The models in these instances are

[1] It is often convenient to refer to the crossover point: parameter values for which 50% of the instances satisfy A.

[2] An implicate is entailed *by* the theory. A prime implicate is one that is not subsumed by another implicate. A unary implicate is just a literal. A binary implicate, for our purposes, is a disjunction of two literals, i. e. a clause of length two.

not randomly distributed, but all lie in a single exponentially large cluster, which moreover admits a short and simple description. These "single cluster instances" are harder to solve by local search, and their emergence at the phase transition seems to be linked to the peak in search cost.

2 Random 3SAT background

By Random 3SAT we mean the uniform fixed-length clause distribution [Mitchell, Selman, & Levesque, 1992]. Ensembles are parameterized by (n, c) where n is the total number of variables, and c is the number of clauses. The c clauses are generated independently by randomly picking 3 variables and negating each variable with probability of 50%. The clause/variable ratio, $\alpha = c/n$, is used as the control parameter.

There are 2^n possible variable assignments, and any one clause is consistent with 7/8 of these. The average number of models, M, per problem instance is then (e. g. [Williams & Hogg, 1993])

$$M = 2^n \left(\frac{7}{8}\right)^c = 2^{n(1\,-\,\alpha/5.19)} \tag{1}$$

Any region with $M < 1$ must contain unsatisfiable instances, so assuming the existence of a satisfiability phase transition forces it to occur at $\alpha \leq 5.19$. Experiments show that the phase transition actually occurs at $\alpha \approx 4.26$ [Crawford & Auton, 1996] giving an average of $2^{0.18n}$ models per instance. There are arguments that come quite close to this empirical value [Williams & Hogg, 1993; Smith, 1994], but so far there is no derivation of the *exact* transition point. In contrast, for random 2SAT the position of the phase transition is known exactly, but is not so interesting because satisfiability is then in **P**. Actually, there appears to be an intriguing link between exact results and the complexity class of the decision problem in that exact results are associated with **P**. The link is often direct, as for 2SAT, but might also be indirect, e. g. the threshold for the existence of Hamiltonian cycles (which is NP-complete) is known exactly in random graphs. However, this threshold "piggy-backs" the transition for 2-connectedness, which is in **P**. As soon as the graph is 2-connected then it almost surely has a Hamiltonian cycle [Bollobás, 1985]. I am not aware of exact threshold results not of one of these types.

To remove unsatisfiable instances from the ensemble, and also to find the prime implicates we used NTAB, a variant of TABLEAU, [Crawford & Auton, 1996; Schrag & Crawford, 1996]. Such systematic searches are computationally expensive and provide the main limit on the range of accessible values of n. Once we have the UPIs for an instance then we can also generate the "residual" theory. This is a mixed 2SAT/3SAT theory over all the residual, or free, variables (those variables not contained in UPIs). The residual clauses are obtained by using the UPIs to simplify each of the original clauses. If an instance has u UPIs then the residual theory has $f = n - u$ residual variables, and, by definition, must be satisfiable and have no UPIs itself.

Since some of the fastest known algorithms are local search algorithms we also study the performance of WSAT [Selman, Kautz, & Cohen, 1994] on the satisfiable instances.

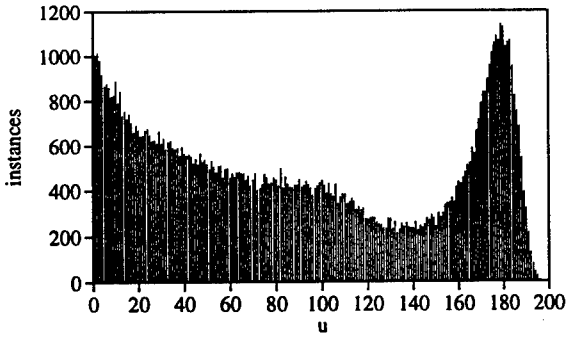


Figure 1: Histogram giving the numbers of instances having u UPIs as a function of u. Obtained from 10^5 satisfiable instances at (n,c)=(200,854).

3 Results for Random 3SAT

In this section we study the "UPI–distribution" as a function of (n, c). After generating instances, we remove those that are unsatisfiable, and for each of the others we find the number of UPIs, u, that it possesses. Counting the number of instances that occur in given ranges of u gives the desired distribution.

Figure 1 gives the UPI–distribution found close to the crossover point for $n = 200$. Figure 2 gives a sample of the results obtained from a "slice" through the phase transition, i. e. varying c at fixed n.

At small c most instances have few UPIs, but the average number of UPIs increases dramatically as we increase c. A peak emerges in the region $170 < u < 190$, and for reasons to be explained later we will call this the "single cluster peak". Conversely, the region $120 < u < 170$ remains consistently underpopulated. Perhaps, figure 2 might best be described as a "sloshing of the UPIs".

We should check that this pattern persists as we increase n, and for this we use the UPI density u/n. Figure 3 shows the effect of increasing n while remaining at the crossover. The distribution pattern seems to persist (except for smaller values of u/n, with which we will not be concerned here). In particular, the position (and indeed existence) of the single-cluster peak is stable at $u/n \approx 0.9$.

4 Interpretation of the UPI slosh

In random graph theory it can be helpful to study the evolution of ensemble properties as we add more edges [Bollobás, 1985]. Here we make some simple arguments in order to study the evolution of the UPI-distribution as we add more clauses. In particular, we propose a partial explanation of the "slosh" of the UPIs as observed in figure 2.

Consider the addition of a single new and randomly selected clause to an existing instance. We want to find the number of new UPIs created as a result of this new clause,

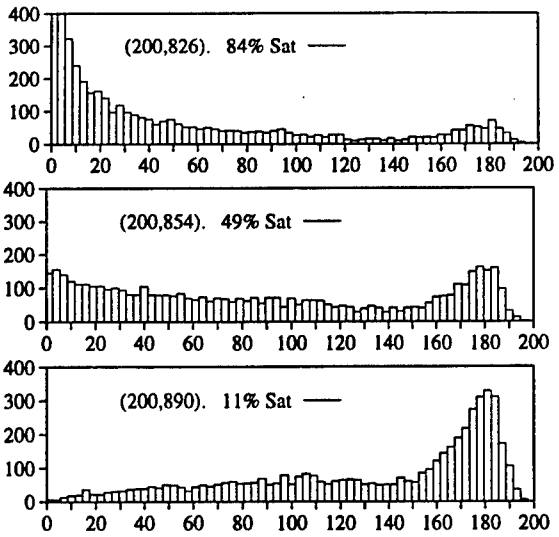


Figure 2: Histograms of the UPI–distribution obtained while traversing the phase transition with n fixed at 200. The legend "x% Sat" means that, at the given (n, c) values, x% of the total ensemble is satisfiable. Each histogram is derived from 5000 satisfiable instances.

and in particular to find how the number of new UPIs varies as a function of the numbers of UPIs and number of binary prime implicates (BPIs) b already present in the instance. We will use figure 4 in order to illustrate the arguments for the crossover point at n=100. It is convenient to define $\lambda = f/n \equiv (1 - u/n)$. Since the new clause is independent of the existing UPIs we have that each literal of the new clause remains free with probability λ, otherwise the UPIs force it true with probability $(1 - \lambda)/2$ or false with probability $(1 - \lambda)/2$. It is then straightforward to find the probabilities of the various fates of the new clause under the existing UPIs. For example, there is a probability of $((1 - \lambda)/2)^3$ that all three literals of the new clause are forced to false rendering the theory unsatisfiable. We must exclude this case (as we only consider satisfiable instances) and then the probability of the new clause reducing to a unary clause, and so directly giving rise to a new UPI is

$$\text{Prob(Unit Clause)} = P(\lambda) = \frac{3\lambda(\frac{1-\lambda}{2})^2}{1 - \frac{1}{8}(1 - \lambda)^3} \tag{2}$$

An example of this function is given in figure 4c.

However, if we happen to create a new UPI then it can give rise to more UPIs by resolution against the b BPIs already present in the theory. Again, since the new clause (and hence initial new UPI) was independent of the existing BPIs it follows that the expected number of BPIs that contain the negation of the new UPI is just $(2b)/(2f)$. So, after a single pass through the BPIs, we will have a total of $(1 + b/f)P(\lambda)$ new UPIs. An example is given in figure 4d based on the empirical data for b.

These latest UPIs can again propagate to cause even more UPIs, but further calculation

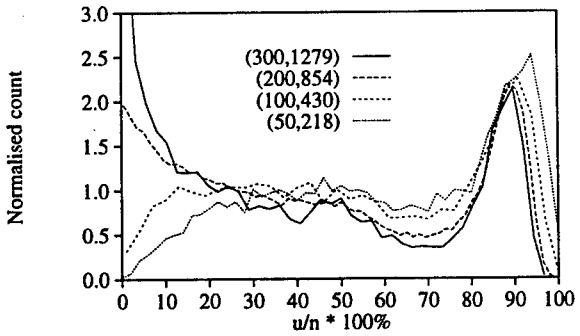


Figure 3: Histogram of UPI–distributions. Here the x-axis is percentage UPIs. The y-axis is the frequency of instances normalized so that a flat distribution would give 1. The (n, c) values are chosen to remain at the crossover point as we increase n.

is hampered by the fact that the latest UPIs need not be be independent of the existing theory. Even with this crude approximation we see from figure 4d that that the effect of a new clause is smallest when u is close to either 0 or n, and largest in the middle of the range.

This allows us to build a picture of the typical evolution of an instance starting in the underconstrained region and adding new clauses while ensuring that the theory does not become unsatisfiable. Being initially underconstrained the instance will start with a small value of u. At first the growth of u will be slow because of the small chance of new clauses generating new UPIs, however once u starts to grow then the peak observed in figure 4d suggests that the growth of u will be rapid until u reaches $u \approx 0.8n$. At this point growth of u will again slow down. This provides a reasonable "first-order" explanation of the deficit of instances in the middle and the accumulation of instances into the single-cluster peak.

5 Clusters

We now look at the nature of instances in the peak observed in the UPI distribution at $u/n \approx 0.9$. The most important property we find is that for instances in the peak, the residual theories (after imposing the UPIs) are very underconstrained, with few clauses per residual variable. This is supported by figure 4b – in the region $u > 0.8n$ there are few BPIs per residual variable. We have also confirmed this by other methods such as direct model counts in the peak region. Hence, these instances have no models outside of the region of the assignment space compatible with the UPIs, but inside this region there are few constraints and the density of models is relatively high: we call such a region a cluster. This cluster of models is compactly described in terms of an assignment to about 90% of the variables and a small number of constraints on the remaining variables. Note that the cluster is small compared to the total assignment space but still exponentially large in n.

Since the peak contains such "single-cluster instances" then it is reasonable to ask what

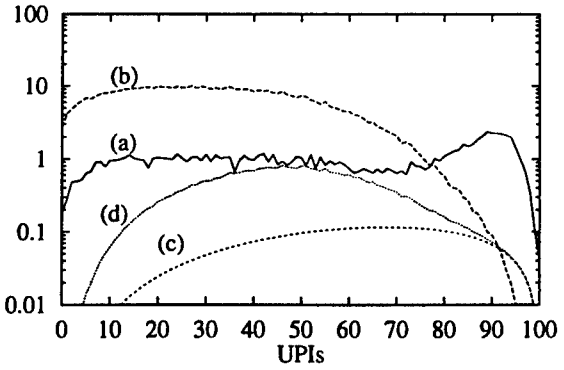


Figure 4: For a sample of 10^4 satisfiable instances at (100,430) we give, as a function of the number of UPIs: (a) the empirical (normalized) frequency of that number of UPIs, (b) the empirical average number of BPIs per free variable, (c) $P(\lambda)$, an estimate of the number of new UPIs directly created from one new clause. (d) as for (c) but also including the effect of propagating the new UPIs *once* through the BPIs.

would happen if we had two such clusters. For simplicity, suppose the clusters were independent, each defined by assignments to a random set of $0.9n$ variables, and with empty residual theories. In this case we only obtain a UPI when the two defining assignments both contain the variable, and also agree on its assignment. Since only 0.9^2n variables occur in both assignments we can expect to get about $0.4n$ UPIs. Adding more clusters would further reduce the number of UPIs, and so we would expect to have few instances with u between these single and double cluster values. Although the assumption of independence of the clusters is probably too strong, the jump of $u \approx 0.9n$ for single cluster instances down to $u \approx 0.4n$ for the hypothesised 2-cluster instances would explain the deficit of instances in the region 0.6–0.8 for u/n seen in figure 1. Also, figure 1 shows some indications of a secondary weak underlying peak at $u/n \approx 0.5$. However, it is clear that further work would be needed to investigate the accuracy of such a multi-cluster interpretation.

6 UPIs and search costs

We now study the effects of the number of UPIs on search cost. Firstly we briefly compare systematic search using NTAB and local search using WSAT. We give results for n=200 (for WSAT we use Maxflips=20000 and noise of 0.5). Figure 5 shows that for NTAB the mean runtime increases fairly smoothly with u, and that over the majority of the range WSAT is faster than NTAB. However, WSAT is badly affected by the single-cluster instances: WSAT spends 70% of its time on the the instances with $\geq$ 75% UPIs, although they form only 27% of the sample. These effects occur for the median as well as the mean, and so we think it is unlikely that they are similar to the extra peaks seen in [Hogg & Williams, 1994; Gent & Walsh, 1994].

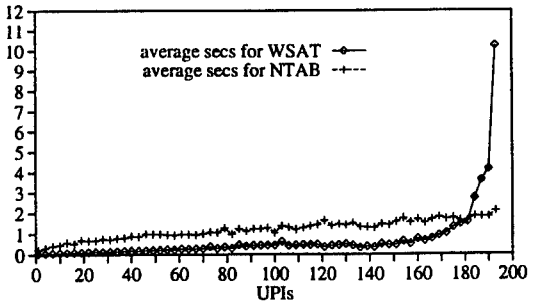


Figure 5: Mean times for NTAB and WSAT to find a solution as a function of the number of UPIs at $(n, c) = (200, 854)$. Based on 10^4 instances. Points above $u = 194$ are omitted due to insufficient instances.

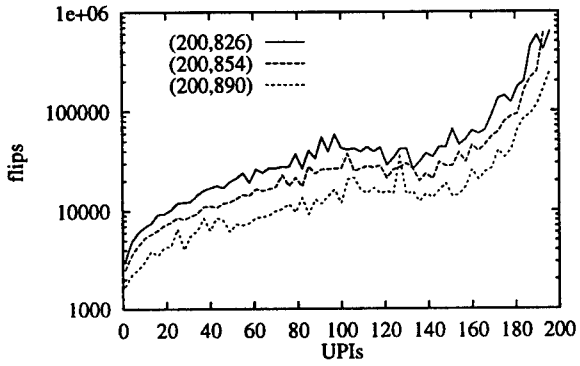


Figure 6: Average flips taken by WSAT plotted against the number of UPIs for various values of c at n=200.

While NTAB is less sensitive to the UPIs it also scales much worse than WSAT (e. g. [Parkes & Walser, 1996]) and so is not the best algorithm for the satisfiable instances (though the results do suggest that it might still remain competitive for those instances that have a lot of UPIs and so are very close to unsatisfiability). Hence, we now restrict ourselves to WSAT.

In figure 6 we look again at the slice through the transition region first considered in figure 2. Across the whole u range it shows that, in the phase transition region, if we fix u, while increasing c, then the theory gets easier. Presumably the extra clauses are helping to direct WSAT towards solutions. Initially this might seem inconsistent with the overall peak of hardness being at crossover point at (n,c)=(200,854). However, we have the competing effect that as we traverse the phase transition region then many instances have a rapid increase in their values for u and so will become much harder. Eventually, the average hardness will be dominated by instances already lying in the single-cluster peak and then the average u does not increase much further, and we will again see the drop in hardness

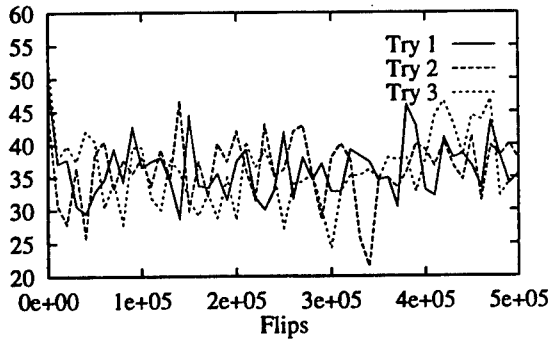


Figure 7: Percentage of UPIs violated against flips for 3 independent tries of WSAT on a hard instance at (350,1491), and having 326 UPIs.

observed at fixed u values. Similar effects are seen with the balance between satisfiable and unsatisfiable instances themselves (Figure 3 of [Mitchell, Selman, & Levesque, 1992]).

Finally we return to the effects of new clauses. Suppose we have a cluster defined by $0.9n$ variables, then there is a probability of at least $0.9^3/8 \approx 0.1$ that a new clause will be violated by all the models in the cluster, converting the cluster into what we call a "failed cluster" (a region much like a cluster but containing no models). Given an instance with two clusters then it is quite possible that a single clause could be responsible for reducing it to a single cluster instance. The impact on search would arise from the fact that the "failed cluster" was initially exponentially large but was all removed by a single clause – local search is then likely to see it as a very attractive region that contains many assignments with just a single violated clause. The resulting exponentially large, but none-the-less false, minimum could easily impede progress.

In figure 7 we look at just one of the hardest single-cluster instances that we found at the crossover at $n = 350$. The progress of WSAT as a function of flips made is measured by the percentage of UPIs correctly matched by the variable assignment. The independent tries consistently settle into a region with many incorrect UPIs (many of the assignments also violate just one clause). The tries vary over the range 30-40% i. e. 10% of the variables which also matches the expected size of the clusters and failed clusters. Hence, this is consistent with the instance having a failed cluster that traps WSAT. Of course, this is just a partial study of just one instance and much more work would be needed to confirm this picture of failed clusters. If true then the exponential size of such failed clusters would also probably adversely affect techniques such as tabu lists that are usually used to escape such false minima. However, the shortness of the description of the clusters (just an assignment to about 90% of the variables) does offer the hope that if it were possible to find such descriptions then they could be used to help escape from the failed clusters.

7 Related work

The effect of the number of models on the cost of local search has been studied in [Clark *et al.*, 1996]. Since instances with large u generally have fewer models our observations on the effect of u are consistent with their confirmation of the expectation that instances with fewer models are harder to solve. The results based on model counting do not seem to show the same clear structure and evolution that we have seen for the UPI-counts. (On the other hand [Clark *et al.*, 1996] did not restrict themselves to Random 3SAT.)

A measure of clause imbalance, Δ, has been suggested as a useful parameter at the phase transition [Sandholm, 1996]. However, Δ itself does not undergo a transition, and is a refinement to α. In contrast, u/n changes abruptly from being nearly zero to being nearly one, and is a refinement to the satisfiability.

Properties of counts of prime implicates are also studied in [Schrag & Crawford, 1996], but by taking averages over instances at a given value of (n,c) rather than looking at the distributions within the ensemble. However, they cover a wider range of parameter values, and also look at longer implicates.

8 Conclusions

The transition region has been described as being "mushy" [Smith, 1994]. We have seen that it can also be described as "slushy": consisting of a mix of frozen instances and fluid instances. The frozen instances have a lot of UPIs but their residual theory is undercon-strained: hence all of their models lie in a single cluster of exponential size but very compact description. The fluid instances have few UPIs. As we traverse the phase transition then more instances freeze. The transition for individual instances is rather fast because instances with a medium number of UPIs are relatively unstable against the addition of extra clauses. Note that we do *not* regard the abrupt change in the number of UPIs as a different phase transition. Instead our view is that there is one transition with many different aspects, of which the changes in satisfiability and UPIs are just the most apparent.

The single-cluster instances have a large and deleterious effect on the local search algorithm WSAT, and often come to dominate the runtimes. It is likely that their emergence at the phase transition is largely responsible for the cost of local search peaking in this region. In contrast, NTAB is very sensitive to the unsatisfiability aspects and less affected by the UPIs. (Presumably, the many aspects of the phase transition mean that every algorithm meets at least some problem, though not necessarily always the same problem.)

We also saw some (weak) evidence for instances with two clusters, or one cluster along with a failed cluster. More work would be needed to see to what extent a multi-cluster description is useful. However, the compactness of the cluster description suggests that it might be of use for knowledge compilation, or for techniques to help local search avoid exponentially large, but false, minima.

Although we have only considered random 3SAT, it could be interesting to make similar investigations for random CSP problems. For example, one could look at how the κ parameter of [Gent *et al.*, 1996] (itself directly related to the average number of models) is

related to the position (or even existence) of the single-cluster peak.

9 Acknowledgments

I am indebted to James Crawford, Joachim Walser, and all the members of CIRL for many helpful comments and discussions.

Effort sponsored by the Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory, Airforce Materiel Command, USAF, under agreements numbered F30602-93-C-0031 and F30602-95-1-0023. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon.

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency (DARPA), the Air Force Research Laboratory, or the U.S. Government.

References

[Bollobás, 1985] Bollobás, B. 1985. *Random graphs.* Academic Press, London.

[Cheeseman, Kanefsky, & Taylor, 1991] Cheeseman, P.; Kanefsky, B.; and Taylor, W. M. 1991. Where the really hard problems are. *Proceedings IJCAI-91* 1.

[Clark *et al.*, 1996] Clark, D.; Frank, J.; Gent, I.; MacIntyre, E.; Tomov, N.; and Walsh, T. 1996. Local search and the number of solutions. In *Proceedings CP-96.*

[Crawford & Auton, 1996] Crawford, J. M., and Auton, L. 1996. Experimental results on the crossover point in random 3-SAT. *Artificial Intelligence* 81:31–57.

[Erdös & Rényi, 1960] Erdös, P., and Rényi, A. 1960. *Publ. Math. Inst. Hung. Acad. Sci.* 5(7).

[Gent & Walsh, 1994] Gent, I., and Walsh, T. 1994. Easy Problems are sometimes Hard. *Artificial Intelligence* 70:335–345.

[Gent *et al.*, 1996] Gent, I. P.; MacIntyre, E.; Prosser, P.; and Walsh, T. 1996. The constrainedness of search. In *Proceedings AAAI-96*, 246–252.

[Hogg & Williams, 1994] Hogg, T., and Williams, C. P. 1994. The hardest constraint problems: a double phase transition. *Artificial Intelligence* 69:359–377.

[Huberman & Hogg, 1987] Huberman, B. A., and Hogg, T. 1987. Phase transitions in artificial intelligence systems. *Artificial Intelligence* 33:155–171.

[Kirkpatrick & Selman, 1994] Kirkpatrick, S., and Selman, B. 1994. Critical behavior in the satisfiability of random boolean expressions. *Science* 264:1297–1301.

[Mitchell, Selman, & Levesque, 1992] Mitchell, D.; Selman, B.; and Levesque, H. 1992. Hard and Easy Distributions of SAT problems. In *Proceedings AAAI-92*, 459–465.

[Parkes & Walser, 1996] Parkes, A. J., and Walser, J. P. 1996. Tuning Local Search for Satisfiability Testing. In *Proceedings AAAI-96*, 356–362.

[Sandholm, 1996] Sandholm, T. W. 1996. A Second Order Parameter for 3SAT. In *Proceedings AAAI-96*, 259.

[Schrag & Crawford, 1996] Schrag, R., and Crawford, J. M. 1996. Implicates and Prime Implicates in Random 3SAT. *Artificial Intelligence* 88:199–222.

[Selman, Kautz, & Cohen, 1994] Selman, B.; Kautz, H. A.; and Cohen, B. 1994. Noise strategies for improving local search. In *Proceedings AAAI-94*, 337–343.

[Smith, 1994] Smith, B. M. 1994. Phase Transition and the Mushy Region in Constraint Satisfaction Problems. In *Proceedings ECAI-94*, 100–104.

[Williams & Hogg, 1993] Williams, C. P., and Hogg, T. 1993. Extending Deep Structure. In *Proceedings AAAI-93*, 152–157.

Appendix H

Supermodels and Robustness*

Matthew L. Ginsberg
CIRL
1269 University of Oregon
Eugene, OR 97403-1269
ginsberg@cirl.uoregon.edu

Andrew J. Parkes
CIRL and CIS Dept.
1269 University of Oregon
Eugene, OR 97403-1269
parkes@cirl.uoregon.edu

Amitabha Roy
CIS Dept.
University of Oregon
Eugene, OR 97403
aroy@cs.uoregon.edu

February 4, 1999

Abstract

When search techniques are used to solve a practical problem, the solution produced is often brittle in the sense that small execution difficulties can have an arbitrarily large effect on the viability of the solution. The AI community has responded to this difficulty by investigating the development of "robust problem solvers" that are intended to be proof against this difficulty.

We argue that robustness is best cast not as a property of the problem solver, but as a property of the solution. We introduce a new class of models for a logical theory, called *supermodels*, that captures this idea. Supermodels guarantee that the model in question is robust, and allow us to quantify the degree to which it is so.

We investigate the theoretical properties of supermodels, showing that finding supermodels is typically of the same theoretical complexity as finding models. We provide a general way to modify a logical theory so that a model of the modified theory is a supermodel of the original. Experimentally, we show that the supermodel problem exhibits phase transition behavior similar to that found in other satisfiability work.

1 Introduction

In many combinatorial optimization or decision problems our initial concern is to find solutions of minimal cost, for example, a schedule with a minimal overall length. In practice, however, such optimal solutions can be very brittle. If anything out of our control goes

*This paper appeared in *Proc. AAAI-98*.

wrong (call this a "breakage"), repairing the schedule might lead to a great increase in its final cost. If breakages are sufficiently common, we might well do better on average to use a suboptimal solution that is more robust. The difficulty with trading optimality for robustness is that robustness is difficult to quantify, and especially difficult to quantify in a practical fashion.

In building definitions that are useful for quantifying robustness, we need to be aware of the requirements of both the users and the producers of robust solutions.

A user of robust solutions might be motivated by two distinct demands on the possibilities for repair:

1. Fast repair: A small set of changes must be repairable in polynomial time.
2. Small repair: The repaired solution should be close to the original model. In other words, it must be possible to repair a small set of changes with another small set of changes.

The condition of fast repair arises, for example, when something goes wrong in a production line and halting the line to perform exponential search might be far too costly. Demanding that small flaws can be addressed with small repairs is also common. A production line schedule might involve many people, each with a different list of tasks for the day. Constantly changing everyone's task list is likely to lead to far too much confusion. The ability to repair flaws with a small number of changes is a goal in itself, independent of the fact that this means repair is also likely to be fast.

As a producer of robust solutions, it might well be helpful if the measure of robustness were independent of the repair algorithm. An algorithm-independent characterization of robustness is useful not only because of its greater simplicity, but because it might support the use of intelligent search methods to find solutions with guaranteed levels of robustness. In contrast, algorithm-dependent notions of robustness imply that the search for robust solutions is likely to reduce to generate and test. This is because partial solutions might not carry enough information to determine whether the repair algorithm will succeed. For example, if we were to use local search for repair, it is already difficult to characterize the repairability of full solutions. Deciding whether a partial solution will extend to a repairable full solution might well be completely impractical. We are not implying that algorithm independence is essential, only that it might be very useful in practice.

This paper introduces the concept of *supermodels* as models that measure inherent degrees of robustness. In essence, a supermodel provides a simple way to capture the requirement that "for all small breakages there exists a small repair;" that repairs are also fast will be seen to follow from this. The supermodel definition also has the advantage that the robustness is inherently a property of the supermodel, and does not rely on assumptions about repair algorithms.

We will also see that despite the simplicity of the supermodel concept, there appears to be a surprisingly rich associated theory. Most importantly, there are many different interrelated classes of supermodel characterized by the amounts of breakage and repair that are allowed. This richness of structure with various degrees of robustness allows us to propose a framework under which robustness can be quantified, thereby supporting an informed tradeoff between optimality and robustness.

The first sections in the paper define supermodels and explore some theoretical consequences of the definition. For satisfiability, finding supermodels is in NP, the same complexity class as that of finding models. We give an encoding that allows us to find a particular kind of supermodel for SAT using standard solvers for SAT. Using this encoding, we explore the existence of supermodels in Random 3SAT, finding evidence for the existence of a phase transition, along with the standard easy-hard-easy transition in search cost.

Overall, the supermodel concept makes the task of finding robust solutions similar to that of finding solutions, rather than necessarily requiring special-purpose search technology of its own.

2 Supermodel Definitions

A first notion of solutions that are inherently robust to small changes can be captured as follows:

> **Definition 1:** An (a, b)-supermodel is a model such that if we modify the values taken by the variables in a set of size at most a (breakage), another model can be obtained by modifying the values of the variables in a disjoint set of size at most b (repair).

The case $a = 0$ means that we never have to handle any breakages, and so all models are also $(0, b)$-supermodels. A less trivial example is a $(1, 1)$-supermodel: This is a model that guarantees that if any single variable's value is changed, then we can recover a model by changing the value of at most one other variable.

We will typically take a and b to be small, directly quantifying the requirement for small repairs. Definition 1 also has a variety of attractive properties. Firstly, if finding models is in NP, and if a is taken to be a constant (independent of the problem size n) then finding (a, b)-supermodels is also in NP. This is because the number of possible breakages is polynomial $O(n^a)$. Secondly, if b is a constant, finding the repair is possible in polynomial time, since there are only $O(n^b)$ possible repairs. These observations are independent of the method used to make the repairs, depending only on the bounded size of the set of possible repairs.

We thus see that with a and b small constants, (a, b)-supermodels quantify our conditions for robustness and do so without worsening the complexity class of the problem (assuming we start in NP or worse).

In practice, the definition needs to be modified because not all variables are on an equal footing. We might not be able to account for some variables changing their value: some breakages might simply be irreparable, while others might be either very unlikely or impossible, and so not worth preparing for. To account for this, we use a "breakage set" that is a subset of the set of all variables, and will only attempt to guarantee robustness against changes of these variables. Similarly, repairs are likely to be constrained in the variables they can change; as an example, it is obviously impossible to modify a variable that refers to an action taken in the past. We therefore introduce a similar "repair set" of variables. We extend Definition 1 to

Definition 2: An (S_1^a, S_2^b)-supermodel is a model such that if we modify the values taken by the variables in a subset of S_1 of size at most a (breakage), another model can be obtained by modifying the values of the variables in a disjoint subset of S_2 of size at most b (repair).

It is clear that an (a, b)-supermodel is simply a (S_1^a, S_2^b)-supermodel in which the breakage and repair sets are unrestricted. We will use the term "supermodel" as a generic term for any (S_1^a, S_2^b)- or (a, b)-supermodel.

Different degrees of robustness correspond to variation in the parameters S_1, S_2, a and b. As we increase the size of the breakage set S_1 or the number of breaks a, the supermodels become increasingly robust. Robustness also increases if we decrease the size of the repair set S_2 or number of repairs b. Supermodels give us a flexible method of returning solutions with certificates of differing but guaranteed robustness. As an example,[1] consider the simple theory $p \vee q$. Any of the three models (p, q), $(\neg p, q)$ and $(p, \neg q)$ is a $(1, 1)$-supermodel. Only the first model, however, is a $(1, 0)$-supermodel. The supermodel ideas correctly identify (p, q) as the most robust model of $p \vee q$.

3 Theory

Let us now restrict our discussion to the case of propositional theories, so that breakage and repair will correspond to flipping the values of variables in the model from true to false or vice versa. We also focus on (a, b)-supermodels as opposed to the more general (S_1^a, S_2^b)-supermodels, and so breakage or repair might involve any variable of the theory.

We shall say that a theory Γ belongs to the class of theories SUPSAT(a, b) if and only if Γ has an (a, b)-supermodel. We will also use SUPSAT(a, b) to refer to the associated decision problem:

SUPSAT(a, b)

Instance: A clausal theory Γ
Question: Is $\Gamma \in$ SUPSAT(a, b) ?

We first prove that SUPSAT(a, b) is in NP for any constants a, b. Given an instance of a theory Γ with n variables, a nondeterministic Turing machine guesses a model and a table of which variables to repair for each set of variables flipped. The table has at most n^a entries, one for each possible possible breakage, and each entry is a list of at most b variables specifying the repair. It is obviously possible to check in polynomial time whether the assignment is a model and that all the repairs do indeed work.

In principle, a supermodel-finding algorithm could produce such a table as output, storing in advance all possible repair tuples. This would take polynomial space $O(n^a b)$ and reduce the time needed to find the repair to be a *constant*, $O(a)$. In practice, however, usage of $O(n^a b)$ memory is likely to be prohibitive.

We also have:

[1] for which we would like to thank Tania Bedrax-Weiss.

Theorem: SUPSAT(1, 1) is NP-hard.

Proof: We reduce SAT to SUPSAT(1, 1).

Let the clausal theory $\Gamma = C_1 \wedge C_2 \ldots \wedge C_m$ over n variables $V = \{x_1 \ldots x_n\}$ be an instance of SAT. We construct an instance of SUPSAT(1, 1) as follows: construct the theory Γ' over $n+1$ variables $V' = \{x_1, x_2 \ldots x_n, \alpha\}$ where

$$\Gamma' = (C_1 \vee \alpha) \wedge (C_2 \vee \alpha) \ldots (C_m \vee \alpha)$$

and α is a new variable not appearing in Γ. We prove that Γ has a model iff Γ' has a (1, 1)-supermodel.

Suppose Γ had a model m. We construct a model for Γ' which will be a (1, 1)-supermodel. Extend the assignment m to an assignment of Γ' by setting α to false. Clearly this assignment satisfies all clauses of Γ'. Suppose now we flip the value of a variable in V'. If we flip the value of some variable in $\{x_1 \ldots x_n\}$, we can repair it by setting $\alpha =$ true. If instead we flip the value of α from false to true, no repair is needed. Hence this assignment is indeed a model of Γ' such that on flipping the value of 1 variable, at most 1 repair is needed. Hence $\Gamma' \in$ SUPSAT(1, 1).

Next, suppose $\Gamma' \in$ SUPSAT(1, 1). By definition, it has a model m. If α is false in m observe that the restriction of m to $V = \{x_1, x_2 \ldots x_n\}$ is a model of Γ. If α is true, flip it to false. Since $\Gamma' \in$ SUPSAT(1, 1) we can repair it by flipping some other variable to get a model where α remains false. Restricting the repaired model to $V = \{x_1, x_2 \ldots x_n\}$ once again gives us a model for Γ. Thus Γ is satisfiable. **QED.**

It follows immediately from the definition that

$$\text{SUPSAT}(a, b) \subseteq \text{SUPSAT}(a, b+1) \tag{1}$$

and

$$\text{SUPSAT}(a+1, b) \subseteq \text{SUPSAT}(a, b) \tag{2}$$

since b repairs suffice for up to $a+1$ breaks, b repairs suffice for up to a breaks. In many cases we can prove that the inclusions in the above supermodel hierarchy are strict.

It is easy to show that the inclusion (1) is strict. i.e. SUPSAT$(a, b) \neq$ SUPSAT$(a, b+1)$. For example the following theory which consists of a chain of $b+2$ variables,

$$(x_1 \rightarrow x_2) \wedge (x_2 \rightarrow x_3) \ldots (x_{b+1} \rightarrow x_{b+2}) \wedge (x_{b+2} \rightarrow x_1)$$

belongs to SUPSAT$(a, b+1)$ − SUPSAT(a, b). The only models of this theory are those with all $b+2$ variables set to true or with all of them set to false. For any set of a flips, we need at most $b+1$ repairs, hence this theory is in SUPSAT$(a, b+1)$. If one variable value is flipped, we need exactly $b+1$ repairs. Since b repairs do not suffice, this theory is not in SUPSAT(a, b).

Using multiple chains and similar arguments, one can prove that the inclusion in (2) is strict whenever $a \leq b$. In general, however, the question of whether or not (2) is strict for all a, b is open.

Equations (1) and (2) induce a hierarchical structure on the set of all satisfiable theories. This gives a rich set of relative "strengths" of robustness with a fairly strong partial order among them.

4 Finding (1,1)-supermodels

We have shown the task of finding (a,b)-supermodels of a SAT problem to be in NP. It should therefore be possible to encode the supermodel requirements on a theory Γ as a new SAT CNF instance Γ_{SM} that is at most polynomially larger than Γ. In this section, we do this explicitly for (1,1)-supermodels in SAT, so that a model for Γ_{SM} has the property that if we are forced to flip any variable i there is another variable j that we can flip in order to recover a model of Γ. In other words, we will show how to construct Γ_{SM} such that Γ has a (1,1)-supermodel if and only if Γ_{SM} has a model. A model of Γ_{SM} will be a supermodel of Γ.

We are working with CNF so $\Gamma = \wedge_a C_a$ is a conjunction of clauses C_a. The basic idea underlying the encoding is to allow the assignment to remain fixed, instead flipping the variables as they appear in the theory.

Thus let Γ_i denote Γ in which all occurrences of variable i have been flipped, and let Γ_{ij} denote Γ in which all occurrences of the variables i and j have been flipped. Denoting the clauses with flipped variables similarly, $\Gamma_i = \wedge_a C_{ai}$ and $\Gamma_{ij} = \wedge_a C_{aij}$.

Now for a model to be a (1,1)-supermodel, if we flip a variable i, one of two conditions must hold. Either the model must be a model of the flipped theory, or there must be some different variable j for which the current model is a model of the doubly flipped theory. Hence, we must enforce

$$\forall i.\ \Gamma_i \vee (\ \exists j.\ j \neq i \wedge \Gamma_{ij}) \tag{3}$$

Converting this to CNF by direct expansion would result in an exponential increase in size, and we therefore introduce new variables c and y that reify the flipped clauses and theories:

$$\begin{aligned} c_{ai} &\longleftrightarrow C_{ai} \\ c_{aij} &\longleftrightarrow C_{aij} \\ y_i &\longleftrightarrow \Gamma_i \\ y_{ij} &\longleftrightarrow \Gamma_{ij} \end{aligned}$$

These definitions are easily converted to a CNF formula Γ_{defs} via

$$\begin{aligned} \neg y_i &= \vee_a \neg c_{ai} \\ \neg y_{ij} &= \vee_a \neg c_{aij} \end{aligned}$$

The supermodel constraint (3) is now

$$\wedge_i (\neg y_i \vee \vee_{j \neq i} \neg y_{ij})$$

which is correctly in CNF. The complete encoding is

$$\Gamma_{SM} = \wedge_i (\neg y_i \vee \vee_{j \neq i} \neg y_{ij}) \wedge \Gamma \wedge \Gamma_{\text{defs}} \tag{4}$$

If the original Γ had n variables and m clauses of length at most k then Γ_{SM} has $O(mn^2)$ variables, and $O(mn^2k)$ clauses of length at most $O(n)$.

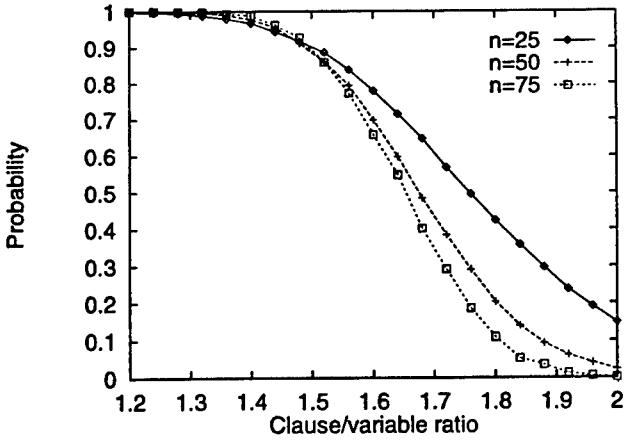


Figure 1: Probability of a Random 3SAT instance having a (1,1)-supermodel.

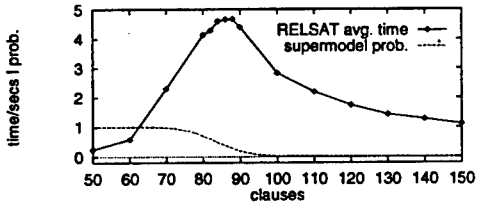


Figure 2: Easy-hard-easy transition at n=50. Time is for `relsat(4)` [Bayardo & Schrag, 1997]. For comparison we give the probability of finding a supermodel.

As an example, consider once again the trivial theory $p \vee q$. The only clause is $C_0 = p \vee q$.

Flipping p, we get $C_{0p} = \neg p \vee q$. Flipping both gives $C_{0pq} = \neg p \vee \neg q$. For the defined variables, we have

$$c_{0p} \leftrightarrow (\neg p \vee q)$$

and similarly, together with $\neg y_p = \neg c_{0p}$ and similarly. The complete theory Γ_{SM} can now be constructed using (4). Note also that the general construction can easily be extended to (S_1^1, S_2^1)-supermodels simply by restricting the allowed subscripts in the c_{ai} and y_i.

Restricting a model of Γ_{SM} to the original variables will produce a (1,1)-supermodel of Γ. Since Γ_{SM} is just another SAT-CNF problem, we can solve it using a standard SAT solver. This solver itself need not know anything about supermodels, and can apply intelligent search techniques that are likely to be significantly more efficient than would be the case if were were to test for robustness in retrospect.

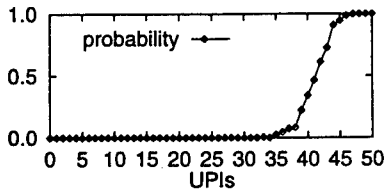


Figure 3: Probability of residual theory having a supermodel, as a function of number of UPIs. Instances are from the phase transition for satisfiability at n=50.

5 Phase Transitions

Phase transition, or "threshold", phenomena are believed to be important to the practical matter of finding solutions [Huberman & Hogg, 1987, and others]. This is in part because of the similarities to optimization: As we change the system, we change from many to relatively few to no solutions, and the cost of finding solutions simultaneously changes from easy to hard to easy again. The average difficulty peaks in the phase transition region, matching the intuition about finding optimal solutions.

In this section, we briefly study the issue of supermodels and phase transitions for the case of Random 3SAT [Mitchell, Selman, & Levesque, 1992]. Instances are characterized by n variables, m clauses, and a clause/variable ratio $\alpha = m/n$. There is strong evidence that for large n, this system exhibits a phase transition at $\alpha \approx 4.2$ [Crawford & Auton, 1996]. Below this value, theories are almost always satisfiable; above it, they are almost always unsatisfiable.

We first consider whether or not SUPSAT(a, b) has similar phase transitions. Using the encoding of the previous section, we studied the empirical probability of Random 3SAT instances having a (1,1)-supermodel. Figure 1 gives the results, leading us to expect a phase transition at $\alpha \approx 1.5$. The apparent SUPSAT$(1, 1)$ transition thus occurs for theories that are very underconstrained. As seen in Figure 2, the time needed to solve the instances undergoes the usual easy-hard-easy transition.

Consider next the possible existence of supermodels at the satisfiability phase transition itself, $\alpha \approx 4.2$. We have just seen that there will almost certainly be no $(1, 1)$-supermodels at this phase transition. We also know that as we approach the transition, the number of prime implicates of the associated theory increases [Schrag & Crawford, 1996], until at the transition itself, we have many instances with large numbers of unary prime implicates (UPIs) [Parkes, 1997]. Any model must respect these UPIs: if a variable in a UPI is changed then no repair is possible. Hence, any variables in UPIs must be excluded from the breakage set. Since flipping the value of a UPI can never be involved in a repair, these variables can also be excluded from the repair set. The simplest choice is thus to look for (S_1^a, S_2^b)-supermodels with

$$S_1 = S_2 = R = (V - \{v | v \text{ or } \neg v \text{ is a UPI}\})$$

Parkes called this set R the *residual variables* of the instance. Looking for an (R^a, R^b)-supermodel is equivalent to looking for an (a,b)-supermodel of the residual theory, which

consists of the constraints remaining on the residual variables after accounting for the UPIs. Figure 3 shows that the residual theories tend to have (1,1)-supermodels in the so-called *single cluster instances*, instances with at least 80% UPIs. For $n = 50$, 80% or more of the variables appear in UPIs some 37% of the time.

The above experiments reflect two extremes. Demanding full (1,1)-supermodels forced us into a very underconstrained region. Conversely, instances from the critically constrained region having many UPIs can still have (S_1^a, S_2^b)-supermodels. In practice, it seems likely that realistic problems will require solutions with intermediate levels of robustness: not so robust as to be able to cater to any possible difficulty, but sufficiently robust as to require some sacrifice in optimality. The framework we have described allows to quantify the tradeoff between robustness and solution cost precisely.

6 Related Work

Since robustness has generally been viewed as a property of the solution *engine* as opposed to a property of the *solutions*, there has been little work on the development of robust solutions to AI problems. Perhaps the most relevant approach has been the attempt to use optimal Markov Decision Processes (MDPs) to find solutions that can recover from likely execution difficulties.

Unfortunately, it appears[2] that the cost of using MDPs to achieve robustness is extreme, in the sense that it is impractical with current technology to solve problems of interesting size. This is to be contrasted to our approach, where the apparent existence of a phase transition suggests that it will be practical to find near-optimal supermodels for problems of practical interest.

Of course, the supermodel approach is solving a substantially easier problem than is the MDP community. We do not (and at this point cannot) consider the differing likelihoods of various failures; a possible breakage is either in the set S_1 or it isn't. We also have no general framework for measuring the probabilistic cost of a solution; we simply require a certain degree of robustness and can then produce solutions that are optimal or nearly so given that requirement. On the other hand, our technology is capable of solving far larger problems and can be applied in any area where satisfiability techniques are applicable, as opposed to the currently restricted domains of applicability of MDPs (planning and scheduling problems, essentially).

A changing environment might also be modeled as a Dynamic Constraint Satisfaction Problem (DCSP) [Dechter & Dechter, 1988]; what we have called a "break" could instead be viewed as the dynamic addition of a unary constraint to the existing theory. The work in DCSPs aiming to prevent the solutions changing wildly from one CSP to the next (e. g. [Verfaillie & Schiex, 1994, and others]) has similar motivations to our requirement for "small repairs", but DCSPs do not supply a way to select solutions to the existing constraints. Supermodels allow us to select the solutions themselves so as to partially guard against future changes of constraints requiring large changes to the solution. Conversely, DCSPs

[2]Steve Hanks, personal communication

can handle changes that are more general than just one set of unary constraints, although we expect that the supermodel idea can be generalized in this direction.

MIXED-CSPs [Fargier, Lang, & Schiex, 1996] allow variables to be controllable (e. g. our flight departure time) or uncontrollable (e. g. the weather). A system is consistent iff any allowed set of values for the uncontrollable variables can be extended to a solution by valuing the controllable variables appropriately. While this has some flavor of preserving the existence of models in the presence of other changes, MIXED-CSPs do not require that the model change be small: No attempt is made to select a model so that nearby worlds have nearby models. On a technical level, this is reflected in the MIXED-CSP consistency check being Π_2^P-complete as opposed to NP-complete for supermodels. Our observations about the phase transitions and reductions to SAT also give us significant practical advantages that are not shared by the MIXED-CSP approach.

7 Conclusions

This paper relies on two fundamental and linked observations. First, robustness should be a property not of the techniques used to solve a problem, but of the solutions those techniques produce. Second, the operational need for solutions that can be modified slightly to recover from small changes in the external environment subsumes the need for solutions for which the repairs can be found quickly. *Supermodels* are a generalization of the existing notion of a model of a logical theory that capture this idea of robustness and that allow us to quantify it precisely.

While the definition of a supermodel is simple, the associated mathematical structure appears to be fairly rich. There is a hierarchy of supermodels corresponding to varying degrees of robustness. Searching for a supermodel is of the same theoretical complexity as solving the original problem, and the experiments on finding supermodels bear this out, revealing a phase transition in the existence of supermodels that is associated with the usual easy-hard-easy transition in terms of computational expense.

Experimental results suggest that finding fully robust supermodels will in general involve substantial cost in terms of the quality of the overall solution. This can be dealt with by considering supermodels that are robust against a limited set of external changes, and we can quantify the expected cost of finding such supermodels as a function of the set of contingencies against which one must guard.

Acknowledgements

This work has been supported by the Air Force Research Laboratory and by the Defense Advanced Research Projects Agency under contracts F30602-95-1-0023 and F30602-97-1-0294. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation hereon.

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA, AFRL, or the U.S. Government.

The authors would like to thank the members of CIRL for many invaluable discussions related to the ideas we have presented.

References

[Bayardo & Schrag, 1997] Bayardo, R. J., and Schrag, R. C. 1997. Using CSP look-back techniques to solve real-world SAT instances. In *Proc. of AAAI-97*, 203–208.

[Crawford & Auton, 1996] Crawford, J. M., and Auton, L. D. 1996. Experimental results on the crossover point in random 3-SAT. *Artificial Intelligence* 81:31–57.

[Dechter & Dechter, 1988] Dechter, R., and Dechter, A. 1988. Belief maintenance in dynamic constraint networks. In *Proc. of AAAI-88*, 37–42.

[Fargier, Lang, & Schiex, 1996] Fargier, H.; Lang, J.; and Schiex, T. 1996. Mixed constraint satisfaction: a framework for decision problems under incomplete knowledge. In *Proc. of AAAI-96*, 175–180.

[Huberman & Hogg, 1987] Huberman, B. A., and Hogg, T. 1987. Phase transitions in artificial intelligence systems. *Artificial Intelligence* 33:155–171.

[Mitchell, Selman, & Levesque, 1992] Mitchell, D.; Selman, B.; and Levesque, H. J. 1992. Hard and easy distributions of SAT problems. In *Proc. of AAAI-92*, 459–465.

[Parkes, 1997] Parkes, A. J. 1997. Clustering at the phase transition. In *Proc. of AAAI-97*, 340–345.

[Schrag & Crawford, 1996] Schrag, R., and Crawford, J. M. 1996. Implicates and prime implicates in Random 3SAT. *Artificial Intelligence* 88:199–222.

[Verfaillie & Schiex, 1994] Verfaillie, G., and Schiex, T. 1994. Solution reuse in Dynamic Constraint Satisfaction Problems. In *Proc. of AAAI-94*, 307–312.

Appendix I

Can Search Play A Role in Practical Applications?[1] [2]

Matthew L. Ginsberg
Brian Drabble
David W. Etherington

CIRL
1269 University of Oregon
Eugene, OR 97403-1269
{ginsberg,drabble,ether}@cirl.uoregon.edu

Abstract

Received wisdom has it that search is ineffective in fielded AI systems; it is argued that realistic problem spaces are simply too large to allow effective search. We suggest that this is because search-based techniques fall prey to early mistakes, making fatal errors sufficiently early in the search that backtrack-based techniques are incapable of correcting them. We describe four general techniques that have been used in practical applications to circumvent this difficulty: limited discrepancy search, schedule packing, squeaky wheel optimization, and relevance-bounded learning. We explain how each technique avoids the early mistakes problem, and describe the impact of the technique on specific problems of practical interest.

[1] This paper appeared in *Proc. of AI Meets the Real World '98*

[2] This work was sponsored in part by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL) under agreement numbers F30602-95-1-0023 and F30602-97-1-0294. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. The views, findings, and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA, AFRL, or the U.S. Government.

1 Introduction

Fielded AI systems tend not to search. The reason is simple: the search spaces involved are astronomically large, and the developers of those systems have learned that adding search capabilities to an application generally does not enhance its performance.

Harvey and Ginsberg [16] show that the ineffectiveness of search-based techniques can be attributed to the fact that heuristics are imperfect, and existing search algorithms are often incapable of backtracking to the point at which a heuristic mistake was made. Harvey and Ginsberg refer to points at which the heuristic errs as *wrong turns*. These wrong turns are particularly devastating near the beginning of the search, since the system may never have time to recover from them. Unfortunately, such "early mistakes" are often likely near the root of the search tree because of the typical lack of information on which to base decisions early in the search. For the large search spaces associated with realistic problems, conventional systematic search algorithms are often incapable of backtracking to the point at which the crucial error occurred.

Instead of abandoning search in the face of this observation, it seems more appropriate to use a search technique that is designed to address the early mistakes problem. In this paper, we discuss four techniques that are designed specifically with this goal in mind. Each of the techniques has already been described elsewhere; our aim here is to present a brief summary of the methods, to cast them specifically in terms of circumventing the early mistakes problem, and then to present results showing the impact of the techniques on problems of practical interest.

The four techniques we will discuss are limited discrepancy search, schedule packing, squeaky wheel optimization, and relevance-bounded learning. We compare the performance of these different techniques on two real world problems: aircraft manufacturing and optical fiber cable assembly. The paper concludes with a summary of the techniques and their applicability.

2 Limited discrepancy search

2.1 Method

Consider the simple search space shown in Figure 1. The tree has been ordered so that the heuristically preferred choice is shown to the left at each point. Depth-first heuristic search will thus examine this space from left to right, visiting the nodes in the order shown in the figure itself. Suppose that node 9 is the unique goal node, so that the heuristic choice of expanding the left node first is wrong at the root, but otherwise correct. The goal is not found until the entire left-hand half of the space has been expanded.

In *limited discrepancy search*, or LDS, the space is examined not from left to right, but in an order that searches the nodes with the fewest "deviations" from the heuristic first [15, 16]. Given that the heuristic choice is always the left-hand one, node 1 in the figure has no heuristic deviations and will be examined first. Nodes 2, 3, 5 and 9 each have one heuristic deviation (one choice of right-hand branch) and will be examined next. Nodes 4, 6, 7, 10, 11 and 13 have two such deviations and are examined next, and so on.

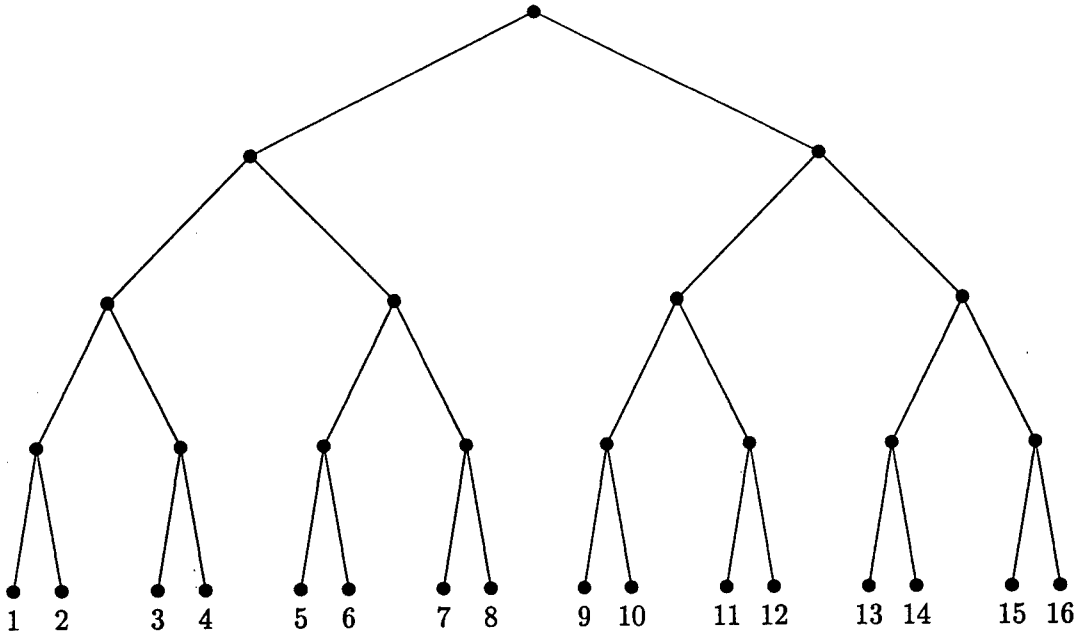


Figure 1: A simple search space

LDS is an iterative algorithm; each iteration repeats the expansions of the previous one.[3] Since the number of nodes expanded in iteration k is generally d times the number expanded in iteration $k-1$ (where d is the depth of the tree), the bulk of the computation is spent on the final iteration and earlier iterations do not significantly affect the run time.

2.2 Early mistakes

The way LDS avoids the early mistakes problem should be clear; in fact, it was during the development of LDS that early mistakes themselves were first recognized and the algorithm is a direct effort to avoid them. By forcing early reconsideration of heuristic choices made near the root of the search tree (or elsewhere), early mistakes can be avoided provided that the heuristic tends, overall, to suggest valid choices.

More recent developments have attempted to focus LDS even more clearly on early mistakes. In *weighted* LDS, or WLDS, the nodes are sorted not simply by counting the number of deviations from the heuristic, but in terms of the heuristic's estimate of the prior probability that the nodes are goal nodes [3]. This allows the algorithm to exploit information indicating how likely it is that the heuristic is correct at any given point. It is possible to show [3] that WLDS properly dominates earlier methods such as iterative sampling [20].

[3] If the depth of the tree is known, some of this repetition can be avoided [19].

2.3 Results

LDS has been applied to the problem of scheduling a variety of activities related to aircraft manufacture. This problem involves sequencing 575 tasks sharing 17 resources and can be found at http://www.NeoSoft.com/~benchmrx.

A 56-day schedule was produced by Mark Ringer (Honeywell) using a simple first fit, interval-based algorithm with no optimization. Agarwal (SAS) has produced a similar result. Both of these results are reported on the previously cited web site, although that site has not been updated recently and Ringer/Agarwal may have improved their times. Colin Bell (University of Iowa) was able to improve on these approaches on a smaller problem but has not posted results for the problem in its entirety. LDS produces schedules somewhat shorter than these, and was found to produce far better schedules still when used in conjunction with schedule packing, which is described in the next section.

3 Schedule packing

3.1 Method

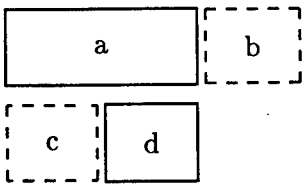
Perhaps the simplest way of avoiding the early mistakes problem is to introduce stochastic elements into the search process itself, thereby ensuring that all decisions are reconsidered with reasonable frequency. In this section, we consider one stochastic search technique known as *schedule packing;* in the next section we consider a generalization known as *squeaky wheel optimization.*

Schedule packing was originally discovered by Barry Fox [11] in the mid-1990's but was not published at that time. Fox subsequently introduced Crawford to a particular set of benchmark problems on which the method proved useful, and collaborated loosely with him on their solution. Schedule packing was rediscovered by Crawford during this time; he extended the method slightly and published it under the name "doubleback optimization" [6].

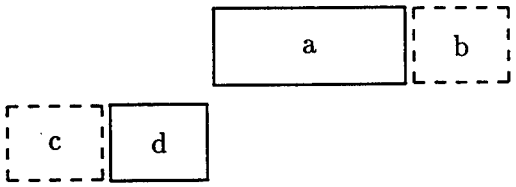
The basic idea is a simple one. Imagine that you have constructed a (not necessarily good) solution to some scheduling problem. It is then straightforward to construct a new, "packed" schedule by selecting an arbitrary ending time that is well past the actual duration of the schedule and then rescheduling each task, starting with the one that finished last in the original and working backwards. Each task is rescheduled to occur as *late* as possible, honoring only precedences and resource conflicts with those tasks that have already been rescheduled. If there are multiple tasks with the same finish time, one of them is selected at random. (This is one source of the stochasticity in the algorithm.) This repacking procedure effectively gives priority to tasks that have occurred late in the original schedule when building the next incarnation.

Having packed the schedule to the right, one can now "repack" it by shifting it to the left as far as possible, this time starting with the task with the earliest start time. Again, tasks that "stick out" are given priority in the hopes that they will find a better position in the resulting schedule. The resulting schedule is frequently shorter than the original. The process can be iterated (remembering the best schedule seen so far) until a sufficiently good

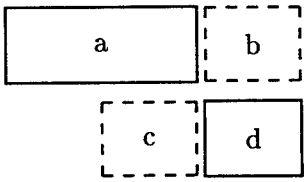
schedule is obtained or time expires. Fox draws an analogy to forcing a box of cereal to settle by turning it over and then back again.



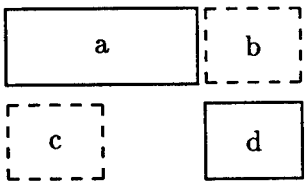
As an example, consider the scheduling problem shown above. The notation is intended to capture the fact that task a is required to precede task b and task c is required to precede task d. The outlines of the boxes indicate that tasks a and d share a resource (and therefore cannot be scheduled simultaneously), as do tasks b and c. If we break the resource contention between a and d by scheduling d first, we obtain the following:



To pack this schedule to the right, we begin by scheduling each job to start as late as possible. This produces:



We then reschedule each task to start as early as possible, scheduling the tasks in the order suggested above. Thus we begin by scheduling a, then c, then either b or d, and then the remaining task. This produces the fully packed schedule shown below.



3.2 Early mistakes

Schedule packing addresses the early mistakes problem in two ways. To understand the first, consider the original scheduling problem in the previous subsection. If we apply schedule packing to the original diagram, we have a choice: we can begin by scheduling task a or

task c. The fact that we make this choice randomly gives the algorithm a stochastic element that helps ensure that all decisions are possibly revisited, independent of their location in the schedule.

Schedule packing also addresses the early mistakes problem directly. In the example, we made a mistake by scheduling task d before task a, even though task a has a following task and task d does not. The packing to the right allows task d to slide past task a, thus identifying the mistake and ensuring that the repacking to the left does not repeat it. In realistic applications, this can potentially move a large part of the schedule.

3.3 Results

On the aircraft manufacturing problem described earlier, a 39-day schedule produced by LDS in combination with schedule packing is the best schedule that has been produced without manual intervention. The best schedule known is a 38-day schedule produced by these two methods in conjunction with some manual focusing of the search. Fox has produced a 41-day schedule using only schedule packing. The combination of LDS as a seed schedule generator and the use of a schedule packing as a post-processing optimizer has proved to be a good match for problems of the complexity of aircraft assembly.

4 Squeaky wheel optimization

4.1 Method

Squeaky wheel optimization (SWO) is a general approach to optimization that was developed by Joslin and Clements [4, 17] through their attempts to better understand and to generalize schedule packing.

In schedule packing, a candidate schedule is first "analyzed" by shifting all of the tasks to the right; the effect of this analysis is to indirectly determine how much each task is contributing to the overall duration of the schedule. Based on this determination, the schedule is then reconstructed, giving priority to the tasks that appear to have the greatest contribution to the overall length.

SWO is a generalization of this technique. It begins by using simple techniques to determine a baseline measure of schedule quality; as an example, a straightforward analysis of precedence conditions can be used to establish a minimum duration for any valid schedule. SWO then takes a specific candidate schedule and examines it to ascertain the extent to which each task contributes to the difference in quality between the baseline schedule and the current one. The schedule is then reconstructed, increasing the priority given to tasks that contribute heavily to the quality difference. In more colorful terms, "The squeaky wheel gets the grease:" in the next iteration, the tasks with increased priority are dealt with earlier and thus get a better pick of the available resources. SWO differs from schedule packing in that the priorities of the various tasks change only slowly from one schedule generation to the next. This appears to allow the method to move a bit more steadily through the search space, gradually improving on the schedule being developed. As with

many stochastic methods, SWO is periodically restarted with a completely fresh schedule so that it can escape from local extrema in the search space.

4.2 Early mistakes

SWO addresses the early mistake problem in the same ways that schedule packing does, by directly and stochastically revisiting problematic decisions. In the direct case, SWO tries to identify tasks leading to difficulties, on the grounds that such tasks may correspond to inopportune early scheduling decisions that need to be revisited. Also, multiple tasks may appear to contribute equally to the quality of an overall schedule, and SWO, like schedule packing, breaks ties randomly. The approach also inherits a stochastic element from the fact that it is periodically restarted with a new random seed schedule.

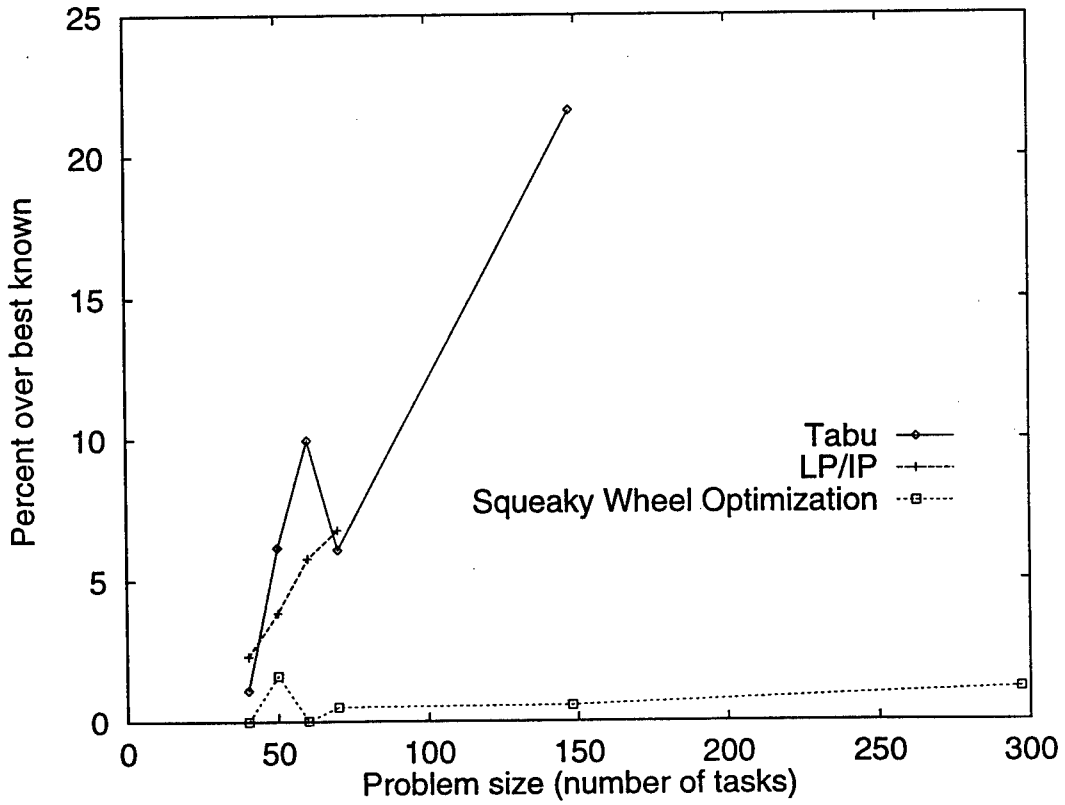


Figure 2: Performance of squeaky wheel optimization

4.3 Results

SWO has been tested on graph coloring problems [17] and on manufacturing problems arising in the construction of fiber-optic cables; we discuss only the latter application here.

Results are shown in Figure 2 comparing the performance of SWO and two well-known

techniques from the operations research (OR) community, tabu search [14], and a linear/integer programming approach combining MINTO [21] and CPLEX [5]. The x-axis gives the number of cables being produced, and the y-axis compares the solution found using the given method to the best solution known.

Previous methods are incapable of finding legal schedules on problems involving more than 150 cables; even on smaller problems, SWO generally finds schedules of substantially higher quality. The best solutions known on the larger problems have been found by using SWO to generate candidate solutions and then refining these solutions using OR techniques.

5 Relevance-bounded learning

5.1 Method

The final approach we will discuss to the early mistake problem is known as *relevance-bounded learning* (RBL) [1, 2]. The basic idea is an extension of truth maintenance [7, 9, 10].

In a truth maintenance system, a reason (or "nogood") is recorded when one is forced to backtrack during the search process. The basic difficulty with this is that search techniques spend the bulk of their time backtracking and it is possible to record a nogood with every inference step. The result of this is that the memory used by the program grows linearly with the runtime, which is obviously impractical on realistic problems.

Ginsberg took a first step toward addressing this difficulty with the introduction of *dynamic backtracking* [12]. The suggestion there was that one only retain learned nogoods as long as they were relevant to the ongoing search, in that they continued to constrain the values of subsequent variables. For example, a nogood such as, "If task 32 takes place at time 46 or later, then task 56 cannot be scheduled after task 133," would remain relevant only as long as task 32 were indeed scheduled to take place at time 46 or later. With the addition of the relevance requirement, Ginsberg showed that truth maintenance techniques would take memory only polynomial in the size of the problem in question (not the run time).

Bayardo and Schrag generalized Ginsberg's ideas, pointing out that instead of requiring that nogoods be relevant, it was sufficient to bound the number of search decisions that would need to be changed for the rule to become relevant again. As an example, suppose that the antecedent in our previous rule was, "If task 32 takes place at time 46 or later, *and task 19 runs on assembly line 12 ...*" As in the previous paragraph, this rule will be relevant only as long as the antecedent is satisfied. But it will be one step "away" from relevance if *either* of the two conditions in the antecedent is satisfied. Bayardo and Schrag observed that the memory requirements of dynamic backtracking would remain polynomial if a relevance bound were established, as opposed to the original requirement that the learned rules actually be relevant at all times ("zero" steps away from relevance, as it were). The performance of the method improved substantially, as more information was learned and retained from early mistakes in the search.

Other authors have also attempted to address this difficulty, perhaps by restricting the length of the learned nogood as opposed to its relevance [8]. This approach tends not to work as well on practical problems because in many cases, the learned nogood applies only

in a very specific context and the size of the antecedent needed to represent that context precludes its inclusion in the size-based scheme. On realistic problems, these large contexts may well be a consequence of choices that are forced by the overall constraints of the problem being solved.

5.2 Early mistakes

RBL addresses the early mistake problem directly. In many cases where a mistake is made early in the search, it is subsequently obvious what has gone wrong. RBL can record the nature of the problem, thereby allowing a direct backtrack to the mistake, skipping over subsequent decisions. RBL also ensures that the same mistake is not repeated subsequently, focusing the search into another area.

5.3 Results

Bayardo and Schrag report on the results of applying RBL to satisfiability problems arising in planning domains [18], circuit diagnosis (from the 1993 DIMACS Challenge, available at ftp://dimacs.rutgers.edu/pub/challenge/satisfiability), and planning, scheduling and circuit synthesis (from the 1996 DIMACS Challenge, http://www.cirl.uoregon.edu/crawford/beijing). On all of these problems, RBL's performance is comparable to that of the best general-purpose satisfiability tools. RBL typically outperforms other systematic methods.

The performance of RBL has not been compared to the performance of special-purpose scheduling or other tools. The basic reason for this is that it is difficult to apply general-purpose satisfiability engines such as RELSAT (Bayardo and Schrag's implementation of RBL) to large problems. Doing so requires translating the problem into a Boolean satisfiability format, typically increasing its size enormously. (As an example, a blocks-world planning problem involving a dozen or so blocks can involve hundreds of thousands of axioms after being converted to an instance of Boolean satisfiability.) If RBL and other satisfiability techniques are to be of practical importance, this difficulty will need to be overcome. Some very recent results of Ginsberg and Parkes [13] suggest that this problem can be met by working not with the ground version of a problem, but directly with the first-order axiomatization that produced it.

6 Conclusion

Search has been of little value on practical problems not because the method itself is inapplicable, but because existing implementations are incapable of correcting mistakes made early in the search. In this paper, we have suggested that it will be possible to address this difficulty using any one of four approaches. Two of these (LDS and RBL) are complete, systematic methods, while the other two (schedule packing and SWO) are stochastic.

Limited discrepancy search addresses the problem by rearranging the order in which the search space is explored. This reordering is correlated with the prior expectation that any particular node is a solution to the problem in question, and has the property that early

decisions are always revisited before large portions of the space have been explored. LDS is a component of the best known solution method on realistic problems related to aircraft manufacture.

Schedule packing is a stochastic method that manipulates candidate solutions to scheduling problems in order to identify those that should have been given priority when the schedule was originally constructed. Assuming that early scheduling mistakes are reflected in the eventual schedule, these mistakes will be identified and can then be avoided when a new schedule is produced. Schedule packing, in conjunction with LDS, leads to the best known solutions on the problems described in the previous paragraph.

Squeaky wheel optimization is an extension and generalization of schedule packing that attempts to identify mistakes generally. Unlike schedule packing, it gradually increases the priority of the decisions associated with these mistakes in an attempt to slowly improve the quality of the solution being produced. On realistic problems from the domain of fiber optic cable construction, SWO is the only known method capable of producing viable solutions to large problems.

Relevance-bounded learning allows a problem-solving system to automatically learn nogoods that can then be used to correct early mistakes and to the avoid their recurrence. Because this technique can only be applied directly to problems that have been expressed as Boolean satisfiability problems, tests of its applicability have thus far been limited. As advances are made in our abilities to translate general problems into this form, RBL can be expected to have an ever increasing role in their solution.

References

[1] R. J. Bayardo and D. P. Miranker. A complexity analysis of space-bounded learning algorithms for the constraint satisfaction problem. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 298–304, 1996.

[2] R. J. Bayardo and R. C. Schrag. Using CSP look-back techniques to solve real-world SAT instances. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, pages 203–208, 1997.

[3] T. Bedrax-Weiss. *Weighted Limited Discrepancy Search.* PhD thesis, University of Oregon, Eugene, OR, 1998.

[4] D. Clements, J. Crawford, D. Joslin, G. Nemhauser, M. Puttlitz, and M. Savelsbergh. Heuristic optimization: A hybrid AI/OR approach. In *Workshop on Industrial Constraint-Directed Scheduling*, 1997.

[5] CPLEX Optimization, Inc. Using the CPLEX callable library and CPLEX mixed integer library, Version 4.0. Technical report, CPLEX Optimization, Inc., 1996.

[6] J. M. Crawford. An approach to resource constrained project scheduling. In *Artificial Intelligence and Manufacturing Research Workshop*, 1996.

[7] J. de Kleer. An assumption-based truth maintenance system. *Artificial Intelligence*, 28:127–162, 1986.

[8] R. Dechter. Enhancement schemes for constraint processing: Backjumping, learning, and cutset decomposition. *Artificial Intelligence*, 41:273–312, 1990.

[9] J. Doyle. A truth maintenance system. *Artificial Intelligence*, 12:231–272, 1979.

[10] J. Doyle. The ins and outs of reason maintenance. In *Proceedings of the Eigth International Joint Conference on Artificial Intelligence*, pages 349–351, 1983.

[11] B. Fox. An algorithm for scheduling improvement by scheduling shifting. Technical Report 96.5.1, McDonnell Douglas Aerospace - Houston, 1996.

[12] M. L. Ginsberg. Dynamic backtracking. *Journal of Artificial Intelligence Research*, 1:25–46, 1993.

[13] M. L. Ginsberg and A. J. Parkes. Search, subsearch and QPROP. In *Fourth International Conference on Principles and Practice of Constraint Programming (PPCP-98)*, Pisa, Italy, 1998. Submitted.

[14] F. Glover and M. Laguna. *Tabu Search.* Kluwer, 1997.

[15] W. D. Harvey. *Nonsystematic Backtracking Search.* PhD thesis, Stanford University, Stanford, CA, 1995.

[16] W. D. Harvey and M. L. Ginsberg. Limited discrepancy search. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 607–613, 1995.

[17] D. Joslin and D. Clements. Squeaky wheel optimization. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, 1998.

[18] H. Kautz and B. Selman. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, 1996.

[19] R. E. Korf. Improved limited discrepancy search. Technical report, UCLA, Los Angeles, CA, 1995.

[20] P. Langley. Systematic and nonsystematic search strategies. In *Artificial Intelligence Planning Systems: Proceedings of the First International Conference*, pages 145–152. Morgan Kaufmann, 1992.

[21] G. Nemhauser, M. Savelsbergh, and G. Sigismondi. MINTO, A mixed integer optimizer. *Operations Research Letters*, 15:47–58, 1994.

Appendix J

Satisfiability Algorithms and Finite Quantification*

Matthew L. Ginsberg

Andrew J. Parkes
CIRL
1269 University of Oregon
Eugene, OR 97403–1269
U.S.A.

February 4, 1999

Abstract

This paper makes three observations with regard to the application of algorithms such as WSAT and RELSAT to problems of practical interest. First, we identify a specific calculation ("subsearch") that is performed at each inference step by any existing satisfiability algorithm. We then show that for realistic problems, the time spent on subsearch can be expected to dominate the computational cost of the algorithm. Finally, we present a specific modification to the representation that exploits the structure of naturally occurring problems and leads to exponential reductions in the time needed for subsearch.

1 Introduction

The last few years have seen extraordinary improvements in the effectiveness of general-purpose Boolean satisfiability algorithms. This work began with the application of WSAT [Selman *et al.*, 1993] to "practical" problems in a variety of domains (generative planning, circuit layout, and others) by translating these problems into propositional logic and then solving them using WSAT.

Although WSAT is unchallenged in its ability to find models for randomly generated satisfiable theories, systematic methods have closed the gap on theories corresponding to practical problems. The algorithm of choice appears to be RELSAT [Bayardo and Schrag, 1997], an extension of an idea from dynamic backtracking [Ginsberg, 1993]. RELSAT is systematic and can therefore deal with both satisfiable and unsatisfiable theories; for theories arising from practical problems, its performance is comparable to WSAT's.

*Effort sponsored by the Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory, Airforce Materiel Command, USAF, under agreement numbered F30602-95-1-0023. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency (DARPA), the Air Force Research Laboratory, or the U.S. Government.

The applicability of propositional algorithms to practical problems is anything but straightforward, however. As the problems become more interesting, their size grows enormously. A single axiom such as

$$\forall xyz.[a(x,y) \wedge b(y,z) \rightarrow c(x,z)] \tag{1}$$

has d^3 ground instances if d is the size of the domain from which x, y and z are taken. Researchers have dealt with this difficulty by buying machines with more memory (much of the recent WSAT work has been done on a machine with 6 *giga*bytes of main memory, making it difficult for this work to be duplicated by others), or by finding clever axiomatizations for which ground theories remain manageably sized [Kautz and Selman, 1998]. In general, memory and cleverness are both scarce resources and a more natural solution will need to be found.

We will call a clause such as (1) *quantified*, and assume throughout that the quantification is universal as opposed to existential, and that the domains of quantification are finite.

Quantified clauses are common in encodings of realistic problems, and these problems have in general been solved by converting quantified clauses to standard propositional formulae. The quantifiers are expanded first (possible because the domains of quantification are finite), and the resulting set of predicates is then "linearized" by relabelling all the atoms so that, for example, $a(2,3)$ might become v_{24}. The number of ground clauses produced is exponential in the number of variables in the quantified clause.

Our primary goal in this paper is to work with the quantified formulation directly, as opposed to its much larger ground translation. Unfortunately, there are significant constant-factor costs incurred in doing so, since each inference step will need to deal with issues involving the bindings of the variables in question. Simply finding the value assigned to $a(2,3)$ might well take two or three times as long as finding the value assigned to the equivalent v_{24}. Finding all occurrences of a given literal can be achieved in the ground case by simple indexing schemes, whereas in the quantified case this is likely to require a unification step. Such routine but essential operations can be expected to significantly slow the cost of every inference undertaken by the system.

The fundamental point of this paper is that while there are costs associated with using quantified axioms, there are significant savings as well. These savings are a consequence of the fact that the basic inference loop of existing boolean satisfiability engines uses an amount of time that scales with the size of the theory; use of quantified axioms can reduce the size of the theory so substantially that the constant-factor costs can be overcome.

We will make this argument in three phases. First, we begin in the next section by describing WSAT and the Davis-Putnam algorithms in detail. In Section 3, we identify a computational subtask that is shared by these and all other existing algorithms. This subtask is NP-complete in a formal sense, and we call it *subsearch* for that reason. We explain why algorithms can be expected to encounter subsearch at each inference step, and show that while subsearch is generally not a problem for randomly generated theories, the subsearch cost will dominate the running time on more realistic instances.

In Section 4, we discuss other consequences of the fact that subsearch is NP-complete. Search techniques can be used to speed the solution of NP-complete problems, and subsearch

is no exception. We show that quantified axiomatizations support the application of simple search techniques to the subsearch problem, and argue that realistic examples are likely to lead to subsearch problems of only polynomial difficulty although existing algorithms such as WSAT solve them exponentially.

Section 5 presents experimental results comparing the performance of WSAT and a generalized version that is capable of dealing with nonground axioms like (1). For small theories, the constant-factor costs of dealing with the variables cause "lifted WSAT" to be some 2–10 times slower than its ground version. As the theories get larger, however, the flip rate of lifted WSAT remains roughly constant while that of ground WSAT drops dramatically. Related and future work are discussed in Sections 6 and 7.

2 Searching for Models

In order to provide a context for the ideas we are about to present, let us begin by giving brief descriptions of the WSAT and Davis-Putnam algorithms.

Procedure 2.1 (WSAT)

```
for i := 1 to MAX-TRIES
  P := a randomly generated truth assignment
  for j := 1 to MAX-FLIPS
    if P is a solution, then return it
    else c := a randomly selected unsatisfied clause
      with probability p, flip a random atom in c
      with probability 1 − p, flip an atom in c that
          produces the fewest newly broken clauses
    end if
  end for
end for
return failure
```

At each iteration, WSAT first checks to see if the problem has been solved. If not, it flips an atom either selected randomly from an unsatisfied clause or selected so as minimize the number of clauses that will change from satisfied to unsatisfied.

Davis-Putnam is best described via a procedure `solve` that accepts as arguments a problem C and a partial assignment P of values to atoms, and returns $\texttt{solve}(C, P)$. It begins with *unit propagation*, a polytime procedure that assigns forced values to atoms. If unit propagation reveals the presence of a contradiction, we return failure. If it turns P into a solution, we return that solution. Otherwise, we pick a branch variable v and try binding it to true and to false in succession. In practice, some effort is made to order the values for v in a way that is likely to lead to a solution quickly.

Procedure 2.2 (Davis-Putnam)

if `unit-propagate`(P) fails, **then** return failure
else set $P :=$ `unit-propagate`(P)
if P is a solution, **then** return it
$v :=$ an atom not assigned a value by P
if `solve`$(C, P \cup \{v = \texttt{true}\})$ succeeds, **then** return it
else return `solve`$(C, P \cup \{v = \texttt{false}\})$

Procedure 2.3 (Unit propagation) In order to compute `unit-propagate`(P):

while there is a currently unsatisfied clause $c \in C$ that
    contains at most one unvalued literal **do**
  if every atom in c is assigned a value by P,
    then return failure
  else $a :=$ the atom in c unassigned by P
    augment P by valuing a so that c is satisfied
  end if
end while
return P

Unit propagation finds unsatisfied clauses containing single unassigned atoms, and values them as required by the clause. Thus if the given problem contained the clause $a \vee \neg b \vee c$ and a had been valued false and b true by P, we would conclude without branching that c must be true if the clause is to be satisfied.

If there were an additional clause $a \vee \neg c$ in the problem, valuing c would value every atom in this clause without satisfying it, and unit propagation would return failure to indicate that a contradiction had been detected.

In the original Davis-Putnam algorithm, we were imprecise regarding the technique used to select the branch variable v when branching was required. Effective implementations pick v to maximize the number of unit propagations that occur after v is valued, since each unit propagation reduces the size of the residual problem. This is typically done by identifying a small number of candidate branch variables v' (generally by counting the number of binary clauses in which each such variable appears), and then computing `unit-propagate`$(P \cup \{v' = \texttt{true}\})$ and `unit-propagate`$(P \cup \{v' = \texttt{false}\})$ to see how many unit propagations actually take place.

3 Subsearch

Each iteration of either the WSAT or Davis-Putnam algorithms involves a search through the original theory for clauses that satisfy some numeric property. In WSAT, when we want to flip the atom a that minimizes the number of newly unsatisfied clauses, we need to count the number of clauses that contain a single satisfied literal and include a in its current sense;

these are the clauses that will become unsatisfied when we flip a.

In unit propagation, used both by the main loop in Davis-Putnam and by the step that selects a branch variable, we need to find the currently unsatisfied clauses that contain a single unvalued literal. Even determining if a solution has been found involves checking to see if any unsatisfied clauses remain.

All of these tasks can be rewritten using the following:

Definition 3.1 *Suppose C is a set of quantified clauses, and P is a partial assignment of values to the atoms in those clauses. We will denote by $S(C, P, u, s)$ the set of ground instances of C that have u literals unvalued by P and s literals satisfied by the assignments in P.*

We will say that the checking problem *is that of determining whether $S(C, P, u, s) \neq \emptyset$. By a* subsearch problem, *we will mean an instance of the checking problem, or the problem of either enumerating $S(C, P, u, s)$ or determining its size.*

Proposition 3.2 *For fixed u and s, the checking problem is NP-complete.*

Proof. Checking is in NP, since a witness that $S(C, P, u, s) \neq \emptyset$ need simply give suitable bindings for the variables in each clause of C.

To see NP-completeness, we assume $u = s = 0$; other cases are not significantly more difficult. We reduce from a binary CSP, producing a single clause C and set of bindings P such that $S(C, P, 0, 0) \neq \emptyset$ if and only if the original problem was satisfiable. The basic idea is that each variable in the constraint problem will become a quantified variable in C.

Suppose that we have a binary CSP Σ with variables $v_1, \ldots, v_n$ and with constraints of the form $(v_{i1}, v_{i2}) \in c_i$, where (v_{i1}, v_{i2}) is the pair of variables constrained by c_i. For each such constraint, we introduce a corresponding binary relation $r_i(v_{i1}, v_{i2})$, and take C to be the single quantified clause $\forall v_1, \ldots, v_n. \vee_i r_i(v_{i1}, v_{i2})$. For the assignment P, we set $r_i(v_{i1}, v_{i2})$ to false for all $(v_{i1}, v_{i2}) \in c_i$, and to true otherwise.

Now note that since P values every instance of every r_i, $S(C, P, 0, 0)$ will be nonempty if and only if there is a set of values for v_i such that every literal in $\vee_i r_i(v_{i_1}, v_{i_2})$ is false. Since a literal $r_i(v_{i_1}, v_{i_2})$ is false just in the case the original constraint c_i is satisfied, it follows that $S(C, P, 0, 0) \neq \emptyset$ if and only if the original theory Σ was satisfiable. ∎

For notational convenience in what follows, suppose that C is a theory and that l is a literal. By C_l we will mean that subset of the clauses in C that include l. If C contains quantified clauses, then C_l will as well; the clauses in C_l can be found by matching the literal l against the clauses in C. A recasting of WSAT in the terms of Definition 3.1 appears at the top of the next column. When selecting the literal to flip, $|S(C_l, P, 0, 1)|$ is the number of ground clauses that will become false when l is flipped ¿from true to false. Just as WSAT appeals to subsearch, so does unit propagation (see middle of next column).

It is important to realize that all we are doing here is introducing new notation; the algorithms themselves are unchanged. It is also important to recognize that algorithmic details such as the settings of noise and restart parameters (for WSAT) or the variable and value choice heuristics (for Davis-Putnam) are irrelevant to the general point that these algorithms depend on solving subsearch problems at each step.

Procedure 3.3 (WSAT)

```
for i := 1 to MAX-TRIES
  P := a randomly generated truth assignment
  for j := 1 to MAX-FLIPS
    if S(C,P,0,0) = ∅, then return P
    else c := a randomly selected clause from S(C,P,0,0)
      with probability p, flip a random atom in c
      with probability 1 − p, flip an atom in c for which
        |S(C_l,P,0,1)| is minimal
    end if
  end for
end for
return failure
```

Procedure 3.4 (Unit propagation)

```
while S(C,P,0,0) ∪ S(C,P,1,0) ≠ ∅ do
  if S(C,P,0,0) ≠ ∅, then return failure
  else select c ∈ S(C,P,1,0)
    augment P so that c is satisfied
  end if
end while
return P
```

In practice, efficient implementations of these algorithms typically compute $S(C,P,u,s)$ once during an initialization phase, and then update it incrementally using a rule such as

$$S(C,P',0,0) = S(C,P,0,0) \cup S(C_{\neg l},P,1,0)$$

if the literal l is changed from unvalued to true. P' here is the partial assignment after the update; P is the assignment before. To compute the number of fully assigned but unsatisfied clauses after the update, we start with the number before, and add newly unsatisfied clauses (unsatisfied clauses previously containing the single unvalued literal $\neg l$).

Reorganizing the computation in this way leads to substantial speedups, since the subsearch problem being solved is no longer NP-complete in the size of C, but only in the size of C_l or $C_{\neg l}$. The fact that these incrementation techniques are essential to the performance of modern search implementations provides further evidence that the performance of these implementations is dominated by the subsearch computation time.

4 Subsearch and quantification

The arguments of the previous section suggest that the running time of existing satisfiability algorithms will be dominated by the time spent solving subsearch problems, since such time is potentially exponential in the size of the subtheory C_l when the literal l is valued, unvalued or flipped. In this section, we discuss the question of how much of a concern this is in practice, and of what (if anything) can be done about it. After all, one of the primary lessons of recent satisfiability research is that problems that are NP-hard in theory tend strongly not to be exponentially difficult in practice.

Let us begin by noting that subsearch is *not* likely to be much of an issue for the randomly generated satisfiability problems that have been the focus of recent research and that have driven the development of algorithms such as WSAT. The reason for this is that if n is the number of clauses in a theory C and v is the number of variables in C, then random problems tend to be difficult only for very specific values of the ratio n/v. For 3-SAT (where every clause in C contains exactly three literals), difficult random problems appear at $n/v \approx 4.2$.

For such a problem, the number of clauses in which a particular literal l appears will be small (on average $3 \times 4.2/2 = 6.3$ for random 3-SAT). Thus the size of the relevant subtheory C_l or $C_{\neg l}$ will also be small, and while subsearch cost still tends to dominate the running time of the algorithms in question, there is little to be gained by applying sophisticated techniques to reduce the time needed to examine a relative handful of clauses.

For realistic problems, the situation is dramatically different. Here is an axiom from a logistics domain:

$$\begin{aligned} &\texttt{at}(o,l,t) \wedge \texttt{flight-time}(l,l',dt) \wedge \\ &\qquad \texttt{between}(t,t',t+dt) \rightarrow \neg\texttt{at}(o,l',t') \end{aligned} \tag{2}$$

This axiom says that if an object o is at location l at time t and it takes time dt to fly from l to l', and t' is between t and $t+dt$, then o cannot be at l' at t'.

A given atom of the form $\texttt{at}(o,l,t)$ will appear in $|t|^2|l|$ clauses of the above form, where $|t|$ is the number of time points or increments and $|l|$ is the number of locations. Even if there are only 100 of each, the 10^6 axioms created seem likely to make computing $S(C_l, P, u, s)$ impractical.

Let us examine this computation in a bit more detail. Suppose that we do indeed have an atom $a = \texttt{at}(O,L,T)$ for fixed O, L and T, and that we are interested in counting the number of unit propagations that will be possible if we set a to true. In other words, we want to know how many instances of (2) will be unsatisfied and have a single unvalued literal after we do so.

Existing implementations, faced with this problem (or an analogous one if WSAT or another approach is used), will note that they need now consider axioms of the form (2) for o, l and t bound and as l', t' and dt are allowed to vary. They examine every axiom of this form and count the number of possible unit propagations.

This approach is taken because existing systems use not quantified clauses such as that of (2), but the set of ground instances of those clauses. Computing $S(C, P, u, s)$ for ground C

involves simply checking each axiom individually; indeed, once the axiom has been replaced by its set of ground instances, no other approach seems possible.

Set against the context of a quantified axiom, however, this seems inappropriate. Computing $S(C, P, u, s)$ for a quantified C by reducing C to a set of ground clauses and then examining each is equivalent to solving the original NP-complete problem by generate and test – and if there is one thing that we can state with confidence about NP-complete problems, it is that generate and test is not an effective way to solve them.

Returning to our example with $\texttt{at}(O, L, T)$ true, we are looking for variable bindings for l', dt and t' such that two of the three literals $\neg\texttt{flight-time}(L, l', dt)$, $\neg\texttt{between}(T, t', T+dt)$ and $\neg\texttt{at}(O, l', t')$ are false, and the third is unvalued. Proposition 3.2 suggests that subsearch will be exponentially hard (with respect to the number of quantifiers) in the worst case, but what is it likely to be like in practice?

In practice, things are going to be much better. Suppose that for some possible destination l', we know that $\texttt{flight-time}(L, l', dt)$ is false for all dt except some specific value Δ. We can immediately ignore all bindings for dt except for $dt = \Delta$, reducing the size of the subsearch space by a factor of $|t|$. If Δ depended on previous choices in the search (aircraft loads, etc.), it would be impossible to perform this analysis in advance and thereby remove the unnecessary bindings in the ground theory.

Pushing this example somewhat further, suppose that Δ is so small that $T + \Delta$ is the time point immediately after T. In other words, $\texttt{between}(T, t', T + \Delta)$ will always be false, so that $\neg\texttt{between}(T, t', T + \Delta)$ will always be true and no unit propagation will be possible for any value of t' at all. We can "backtrack" away from the unfortunate choice of l' in our (sub)search for variable bindings for which unit propagation is possible. Such backtracking is not supported by the generate-and-test subsearch philosophy used by existing implementations.

This sort of computational savings is likely to be possible in general. For naturally occurring theories, most of the atoms involved are likely to be either unvalued (because we have not yet managed to determine their truth values) or false (by virtue of the closed-world assumption, if nothing else). Domain constraints will typically be of the form $a_1 \wedge \cdots \wedge a_k \rightarrow l$, where the premises a_i are atoms and the conclusion l is a literal of unknown sign. Unit propagation (or other likely instances of the subsearch problem) will thus involve finding a situation where at most one of the a_i is unvalued, and the rest are true. If we use efficient data structures to identify those instances of relational expressions that are true, it is not unreasonable to expect that most instances of the subsearch problem will be soluble in time polynomial in the length of the clauses involved, as opposed to exponential in that length.

5 Experimental results

To examine the impact of our ideas in an experimental setting, we compared the performance of ground and lifted solvers on problems of a variety of sizes from a simple logistics planning domain [Kautz and Selman, 1998, and others].

The domain consists of objects and airplanes; the locations are given as a function of time by means of predicates such as $\texttt{aboard}(o, a, t)$ for an object o, airplane a and time t.

There are also consistency axioms such as

$$a \neq b \rightarrow \neg\texttt{aboard}(o, a, t) \vee \neg\texttt{aboard}(o, b, t)$$

saying that an object can be in only one plane at a time. The only actions are loading or unloading objects onto planes and flying the planes. Unlike the example of the last section, all flights are assumed to be of unit duration.

Actions can take place in parallel and are interpreted as restricting possible changes of state. For example, since loading or unloading an object requires the relevant plane to be at the airport we require

$$\texttt{at}(o, c, t) \wedge \texttt{aboard}(o, a, t+1) \rightarrow \texttt{plane-at}(a, c, t) \tag{3}$$

The experiments themselves used WSAT to solve problems that use $n-1$ aircraft to redistribute n objects that were initially located at random among $n+1$ cities. In both the initial and goal states, no two objects are in the same city. Since exactly one plane must be used twice (and parallel actions are permitted), the length of optimal plans is independent of the value of n.

Our goal in the experiments is not to measure time to solution, but the basic step rate of the underlying search engine. The actual steps taken by the solvers should be (and are) identical; it is the search rate that is directly impacted by the costs and savings of the lifted representation. Since WSAT is known to spend most of its time in regions of the search space that contain only a few unsatisfied clauses, we actually presented WSAT (and its lifted analog) with unsatisfiable versions of the above problem, requiring that the schedule be complete using one fewer action than the minimum. This allowed us to measure the flip rates of both engines accurately. We also simplified the theories before presenting them to the solvers, propagating unit literals and unit propagating to completion. Since this produced smaller theories (although still of asymptotic size $10n^2$ variables and $11n^3$ axioms), it could be expected to reduce the impact of the subsearch idea while leaving the constant factors unchanged, biasing our experimental results in favor of ground axiomatizations as opposed to lifted ones. All experiments were run on a 200 MHz Pentium Pro with 64MB of memory running Linux.

Let us first consider the time taken to initialize WSAT by counting the numbers of unsatisfied clauses after the initial variable assignment. Figure 1 presents initialization times for both ground WSAT and a general-purpose lifted solver. The lifted solver takes quantified clauses as input and solves the subsearch problems efficiently.

Except for the region $n < 10$, ground WSAT scales as $O(n^3)$, predictably proportional to the number of clauses. WSAT's anomalously bad scaling for $n < 10$ appears to correspond to the ground theory's rapid growth exceeding the size of the CPU's on-board memory cache.

When we solve the subsearch problem efficiently, the scaling drops ¿from $O(n^3)$ to $O(n^2)$. Once again, this is to be expected. One of the axioms that dominates the runtime is (3), which can be rewritten as

$$\forall otca. \neg\texttt{at}(o, c, t) \vee \neg\texttt{aboard}(o, a, t+1) \vee \texttt{plane-at}(a, c, t)$$

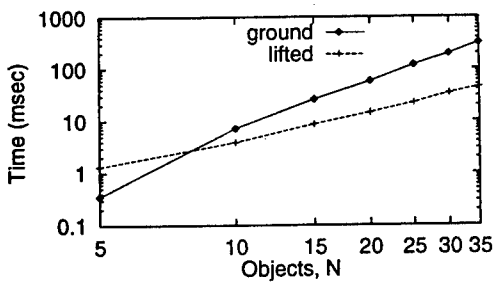


Figure 1: CPU time needed for initialization in WSAT

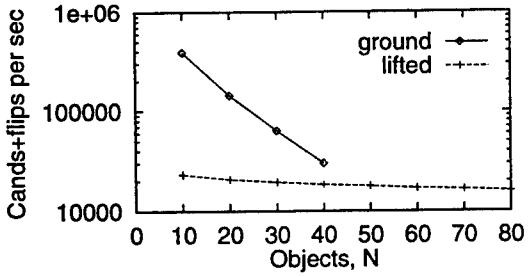


Figure 2: Candidates considered for flipping, and flips themselves, per second. The ground line terminates because the physical memory of the machine is exhausted.

The subsearch for unsatisfied clauses can be terminated whenever $\mathtt{at}(o, c, t)$ is false. This pruning is likely to be common (since objects should only be in one city) and each such pruning avoids the need to consider each plane, so the subsearch involving this axiom is reduced from size *otca* to size *otc*. The former is of size $O(n^3)$, while *otc* is only of size $O(n^2)$ (recall that the number of time points is independent of n). Solving the subsearch problem efficiently improves the complexity of initialization, making it a clear win over the ground case.

We next consider flip performance. After selecting an unsatisfied clause, WSAT must select a specific literal from that clause to flip. The selection procedure consumes a significant fraction of the CPU time, and this fraction increases as the clauses involved become longer. Since this would somewhat obscure the scaling results, the rate used to measure performance is the sum of the rate at which candidates are considered and the rate at which they are actually flipped. Results are in Figure 2.

Ground WSAT initially outperforms the lifted version but the difference drops rapidly as the size of the problem increases. The ground line terminates at $n = 40$ because at that point memory usage, growing as $O(n^3)$ for the ground solver, has increased to 70MB, and thus exceeds physical memory. The actual runtime becomes far longer than the CPU time would indicate. In contrast, the lifted solver shows only a gradual loss of performance.

Extrapolating the lines would suggest that even if it were possible to run the ground solver at larger sizes it would still be outperformed by the lifted solver.

This is in fact somewhat surprising, because the axiomatization used to encode the domain (omitted for reasons of space) has a structure that should show no gain from pruning. We believe that the better scaling of the lifted solver is due to the physical properties of the computer memory system. For the lifted solver, the problem size remains proportional to the number of variables, which is $O(n^2)$. The problem remains smaller and so makes better use of the cache: the ground solver is more likely to have to access clauses in main memory, a far slower process. This suggests that the constant factor overheads associated with lifting can be mitigated or even overcome by hardware advantages alone. Had the underlying theory allowed efficient solution of subsearch problems, we could expect even better scaling properties.

6 Related Work

Other researchers have also examined the problems inherent in dealing with "global" constraints that are quantified over time or objects. They have typically assumed that the constraints involve many variables (polynomially many in the size of the problem), and that the constraints are "composite" in that they can be expressed using a large number of simpler constraints.

The composite nature of global constraints is most clear in constraint programming, where global constraints are often used because they can be implemented more efficiently than the equivalent set of more primitive constraints. A typical example is an `alldifferent` constraint on a set of variables X_i, which is equivalent to the $O(n^2)$ separate inequalities $\forall ij.[i = j \vee X_i \neq X_j]$. The composite form can be implemented to run in time $O(n \log n)$. As an example, if all of the X_i are bound, we can sort their values to check for duplicates.

Lifted constraints are also composite, corresponding to the set of their ground instances. As with global constraints, we see implementation advantages in that the composite, lifted form can be implemented to run faster then the equivalent set of simpler, ground constraints.

There are major differences between our work and that of the global constraints community, however. These include the basic complexity of the constraints: lifted constraints are NP-complete with respect to the number of universal quantifiers, while global constraints such as `alldifferent` are in general polytime, corresponding to a polynomial number of primitive constraints. Checking an `alldifferent` constraint via enumeration involves only a polynomial number of tests (and does not seem like search); checking a lifted constraint via enumeration involves exponentially many tests (and does indeed seem like search). Perhaps more importantly, lifted constraints do not involve extensions beyond first-order syntax, and can therefore be combined effectively with other constraints in ways that `alldifferent` cannot be.

Finally, our methods are hardly the only way to implement search on large problems; specialized code can handle domains of very large size. Our approach has the advantage of generality (and hence of maintainability). We believe it also illuminates general issues that would otherwise be obscured in the specialized code itself.

7 Conclusion and future work

Satisfiability algorithms have generally been developed against the framework provided by either random instances or, worse still, no unbiased source of instances at all. The algorithms themselves have thus tended to ignore problem features that dominate the computational requirements when they are applied to real problems.

Principal among these appears to be *subsearch*, arising from the fact that most computational tasks involving realistic problems are NP-complete in the size of the quantified theory. We have argued that while subsearch is NP-complete in the size of the underlying theory in principle, it is unlikely to be so in practice although existing satisfiability algorithms do not exploit this. We described the reasons for this, and reported on the performance of a generalization of WSAT that is capable of dealing with quantified theories. This generalization runs only modestly more slowly than the original WSAT on randomly generated theories, but substantially more quickly on the large theories associated with naturally occurring problems. The generalization is also capable of solving problem instances for which ground WSAT would likely exhaust the physical memory available.

The most obvious extension to our work involves applying it to a systematic search engine. This should be straightforward for algorithms that do not involve any learning of nogoods. However, the most effective systematic algorithms do involve such learning, and exploiting lifting in this context appears to be more challenging than the WSAT work. We need to deal with the fact that the nogoods learned as the algorithm proceeds should be not the ground result of resolving ground database instances, but instead the result of unifying and then resolving the original quantified axioms. This leads to a variety of technical issues that have not yet been solved, and upon which we will report in a subsequent paper.

References

[Bayardo and Schrag, 1997] R. J. Bayardo and R. C. Schrag. Using CSP look-back techniques to solve real-world SAT instances. In *Proceedings AAAI-97*. AAAI Press.

[Ginsberg, 1993] M. L. Ginsberg. Dynamic backtracking. *Journal of Artificial Intelligence Research*, 1:25–46, 1993.

[Kautz and Selman, 1998] H. Kautz and B. Selman. BLACKBOX: A new approach to the application of theorem proving to problem solving. In *Proceedings AIPS-98*. AAAI Press.

[Selman *et al.*, 1993] B. Selman, H. A. Kautz, and B. Cohen. Local search strategies for satisfiability testing. In *Proceedings 1993 DIMACS Workshop on Maximum Clique, Graph Coloring, and Satisfiability*, 1993.

•U.S. GOVERNMENT PRINTING OFFICE: 1999-610-130-81117

DISTRIBUTION LIST

addresses	number of copies
DR. NORTHRUP FOWLER AFRL/IFT 525 BROOKS ROAD ROME, NY 13441-4505	10
UNIVERSITY OF OREGON OFFICE OF RESEARCH & SPONSORED PROG 5219 UNIVERSITY OF OREGON EUGENE OR 97403-5219	5
AFRL/IFOIL TECHNICAL LIBRARY 26 ELECTRONIC PKY ROME NY 13441-4514	1
ATTENTION: DTIC-OCC DEFENSE TECHNICAL INFO CENTER 8725 JOHN J. KINGMAN ROAD, STE 0944 FT. BELVOIR, VA 22060-6218	2
DEFENSE ADVANCED RESEARCH PROJECTS AGENCY 3701 NORTH FAIRFAX DRIVE ARLINGTON VA 22203-1714	1
ATTN: NAN PFRIMMER IIT RESEARCH INSTITUTE 201 MILL ST. ROME, NY 13440	1
AFIT ACADEMIC LIBRARY AFIT/LDR, 2950 P.STREET AREA B, BLDG 642 WRIGHT-PATTERSON AFB OH 45433-7765	1
AFRL/MLME 2977 P STREET, STE 6 WRIGHT-PATTERSON AFB OH 45433-7739	1

AFRL/HESC-TDC 1
2698 G STREET, BLDG 190
WRIGHT-PATTERSON AFB OH 45433-7604

ATTN: SMDC IM PL 1
US ARMY SPACE & MISSILE DEF CMD
P.O. BOX 1500
HUNTSVILLE AL 35807-3801

COMMANDER, CODE 4TL000D 1
TECHNICAL LIBRARY, NAWC-WD
1 ADMINISTRATION CIRCLE
CHINA LAKE CA 93555-6100

CDR, US ARMY AVIATION & MISSILE CMD 2
REDSTONE SCIENTIFIC INFORMATION CTR
ATTN: AMSAM-RD-OB-R, (DOCUMENTS)
REDSTONE ARSENAL AL 35898-5000

REPORT LIBRARY 1
MS P364
LOS ALAMOS NATIONAL LABORATORY
LOS ALAMOS NM 87545

ATTN: D'BORAH HART 1
AVIATION BRANCH SVC 122.10
FOB10A, RM 931
800 INDEPENDENCE AVE, SW
WASHINGTON DC 20591

AFIWC/MSY 1
102 HALL BLVD, STE 315
SAN ANTONIO TX 78243-7016

ATTN: KAROLA M. YOURISON 1
SOFTWARE ENGINEERING INSTITUTE
4500 FIFTH AVENUE
PITTSBURGH PA 15213

USAF/AIR FORCE RESEARCH LABORATORY 1
AFRL/VSOSA(LIBRARY-BLDG 1103)
5 WRIGHT DRIVE
HANSCOM AFB MA 01731-3004

ATTN: EILEEN LADUKE/D460 1
MITRE CORPORATION
202 BURLINGTON RD
BEDFORD MA 01730

OUSD(P)/DTSA/DUTD 1
ATTN: PATRICK G. SULLIVAN, JR.
400 ARMY NAVY DRIVE
SUITE 300
ARLINGTON VA 22202

DR JAMES ALLEN 1
COMPUTER SCIENCE DEPT/BLDG RM 732
UNIV OF ROCHESTER
WILSON BLVD
ROCHESTER NY 14627

DR YIGAL ARENS 1
USC-ISI
4676 ADMIRALTY WAY
MARINA DEL RAY CA 90292

DR MARIE A. BIENKOWSKI 1
SRI INTERNATIONAL
333 RAVENSWOOD AVE/EK 337
MENLO PRK CA 94025

DR MARK S. BODDY 1
HONEYWELL SYSTEMS & RSCH CENTER
3660 TECHNOLOGY DRIVE
MINNEAPOLIS MN 55418

DR PIERO P. BONISSONE 1
GE CORPORATE RESEARCH & DEVELOPMENT
BLDG K1-RM 5C-32A
P. O. BOX 8
SCHENECTADY NY 12301

DR MARK BURSTEIN 1
BBN SYSTEMS & TECHNOLOGIES
10 MOULTON STREET
CAMBRIDGE MA 02138

DR THOMAS L. DEAN 1
BROWN UNIVERSITY
DEPT OF COMPUTER SCIENCE
P.O. BOX 1910
PROVIDENCE RI 02912

DR WESLEY CHU 1
COMPUTER SCIENCE DEPT
UNIV OF CALIFORNIA
LOS ANGELES CA 90024

DR PAUL R. COHEN 1
UNIV OF MASSACHUSETTS
COINS DEPT
LEDERLE GRC
AMHERST MA 01003

DR JON DOYLE 1
LABORATORY FOR COMPUTER SCIENCE
MASS INSTITUTE OF TECHNOLOGY
545 TECHNOLOGY SQUARE
CAMBRIDGE MA 02139

DR. BRIAN DRABBLE 1
CIRL, 1269
UNIVERSITY OF OREGON
EUGENE, OR 97403

MR. SCOTT FOUSE 1
ISX CORPORATION
4353 PARK TERRACE DRIVE
WESTLAKE VILLAGE CA 91361

DR MICHAEL FEHLING 1
STANFORD UNIVERSITY
ENGINEERING ECO SYSTEMS
STANFORD CA 94305

RICK HAYES-ROTH 1
CIMFLEX-TEKNOWLEDGE
1810 EMBARCADERO RD
PALO ALTO CA 94303

MS. YOLANDA GIL 1
USC/ISI
4676 ADMIRALTY WAY
MARINA DEL RAY CA 90292

MR. MARK A. HOFFMAN 1
ISX CORPORATION
1165 NORTHCHASE PARKWAY
MARIETTA GA 30067

DR RON LARSEN 1
NAVAL CMD, CONTROL & OCEAN SUR CTR
RESEARCH, DEVELOP, TEST & EVAL DIV
CODE 444
SAN DIEGO CA 92152-5000

DR CRAIG KNOBLOCK 1
USC-ISI
4676 ADMIRALTY WAY
MARINA DEL RAY CA 90292

DR JOHN LOWRENCE 1
SRI INTERNATIONAL
ARTIFICIAL INTELLIGENCE CENTER
333 RAVENSWOOD AVE
MENLO PARK CA 94025

DR. ALAN MEYROWITZ 1
NAVAL RESEARCH LABORATORY/CODE 5510
4555 OVERLOOK AVE
WASH DC 20375

ALICE MULVEHILL 1
BBN
10 MOULTON STREET
CAMBRIDGE MA 02238

DR ROBERT MACGREGOR 1
USC/ISI
4676 ADMIRALTY WAY
MARINA DEL REY CA 90292

DR DREW MCDERMOTT 1
YALE COMPUTER SCIENCE DEPT
P.O. BOX 2158, YALE STATION
51 PROPSPECT STREET
MEW HAVEN CT 06520

DR DOUGLAS SMITH 1
KESTREL INSTITUTE
3260 HILLVIEW AVE
PALO ALTO CA 94304

DR. AUSTIN TATE 1
AI APPLICATIONS INSTITUTE
UNIV OF EDINBURGH
80 SOUTH BRIDGE
EDINBURGH EH1 1HN - SCOTLAND

DIRECTOR 1
DARPA/ITO
3701 N. FAIRFAX DR., 7TH FL
ARLINGTON VA 22209-1714

DR STEPHEN F. SMITH 1
ROBOTICS INSTITUTE/CMU
SCHENLEY PRK
PITTSBURGH PA 15213

DR JONATHAN P. STILLMAN 1
GENERAL ELECTRIC CRD
1 RIVER RD, RM K1-5C31A
P. O. BOX 8
SCHENECTADY NY 12345

DR EDWARD C.T. WALKER 1
BBN SYSTEMS & TECHNOLOGIES
10 MOULTON STREET
CAMBRIDGE MA 02138

DR BILL SWARTOUT 1
USC/ISI
4676 ADMIRALTY WAY
MARINA DEL RAY CA 90292

GIO WIEDERHOLD 1
STANFORD UNIVERSITY
DEPT OF COMPUTER SCIENCE
438 MARGARET JACKS HALL
STANFORD CA 94305-2140

DR KATIA SYCARA/THE ROBOTICS INST 1
SCHOOL OF COMPUTER SCIENCE
CARNEGIE MELLON UNIV
DOHERTY HALL RM 3325
PITTSBURGH PA 15213

DR DAVID E. WILKINS 1
SRI INTERNATIONAL
ARTIFICIAL INTELLIGENCE CENTER
333 RAVENSWOOD AVE
MENLO PARK CA 94025

DR. PATRICK WINSTON
MASS INSTITUTE OF TECHNOLOGY
RM NE43-817
545 TECHNOLOGY SQUARE
CAMBRIDGE MA 02139 1

DR STEVE ROTH
CENTER FOR INTEGRATED MANUFACTURING
THE ROBOTICS INSTITUTE
CARNEGIE MELLON UNIV
PITTSBURGH PA 15213-3890 1

DR YOAV SHOHAM
STANFORD UNIVERSITY
COMPUTER SCIENCE DEPT
STANFORD CA 94305 1

MR. LEE ERMAN
CIMFLEX TECKNOWLEDGE
1810 EMBARCARDERO RD
PALO ALTO CA 94303 1

DR MATTHEW L. GINSBERG
CIRL, 1269
UNIVERSITY OF OREGON
EUGENE OR 97403 1

MR JEFF GROSSMAN, CO
NCCOSC RDTE DIV 44
5370 SILVERGATE AVE, ROOM 1405
SAN DIEGO CA 92152-5146 1

DR ADELE E. HOWE
COMPUTER SCIENCE DEPT
COLORADO STATE UNIVERSITY
FORT COLLINS CO 80523 1

DR LESLIE PACK KAELBLING
COMPUTER SCIENCE DEPT
BROWN UNIVERSITY
PROVIDENCE RI 02912 1

DR SUBBARAO KAMBHAMPATI
DEPT OF COMPUTER SCIENCE
ARIZONA STATE UNIVERSITY
TEMPE AZ 85287-5406 1

DR CARLA GOMES 1
AFRL/IFTB
525 BROOKS RD
ROME NY 13441-4505

DR KAREN MYERS 1
AI CENTER
SRI INTERNTIONAL
333 RAVENSWOOD
MENLO PARK CA 94025

DR MARTHA E POLLACK 1
DEPT OF COMPUTER SCIENCE
UNIVERSITY OF PITTSBURGH
PITTSBURGH PA 15260

DR RAJ REDDY 1
SCHOOL OF COMPUTER SCIENCE
CARNEGIE MELLON UNIVERSITY
PITTSBURGH PA 15213

DR EDWINA RISSLAND 1
DEPT OF COMPUTER & INFO SCIENCE
UNIVERSITY OF MASSACHUSETTS
AMHERST MA 01003

DR MANUELA VELOSO 1
CARNEGIE MELLON UNIVERSITY
SCHOOL OF COMPUTER SCIENCE
PITTSBURGH PA 15213-3891

DR DAN WELD 1
DEPT OF COMPUTER SCIENCE & ENG
MAIL STOP FR-35
UNIVERSITY OF WASHINGTON
SEATTLE WA 98195

MR JOE ROBERTS 1
ISX CORPORATION
4301 N FAIRFAX DRIVE, SUITE 301
ARLINGTON VA 22203

DR TOM GARVEY 1
SRI INTERNATIONAL
ARTIFICIAL INTELLIGENCE CENTER
333 RAVENSWOOD AVE
MENLO PARK CA 94025

DIRECTOR
DARPA/ISO
3701 NORTH FAIRFAX DRIVE
ARLINGTON VA 22203-1714 1

OFFICE OF THE CHIEF OF NAVAL RSCH
CODE 311
800 N. QUINCY STREET
ARLINGTON VA 22217 1

DR GEORGE FERGUSON
UNIVERSITY OF ROCHESTER
COMPUTER STUDIES BLDG, RM 732
WILSON BLVD
ROCHESTER NY 14627 1

DR STEVE HANKS
DEPT OF COMPUTER SCIENCE & ENG'G
UNIVERSITY OF WASHINGTON
SEATTLE WA 98195 1

DR CHRISTOPHER OWENS
GTE
10 MOULTON ST
CAMBRIDGE MA 02138 1

DR JAIME CARBONNEL
THE ROBOTICS INSTITUTE
CARNEGIE MELLON UNIVERSITY
DOHERTY HALL, ROOM 3325
PITTSBURGH PA 15213 1

DR NORMAN SADEH
THE ROBOTICS INSTITUTE
CARNEGIE MELLON UNIVERSITY
DOHERTY HALL, ROOM 3315
PITTSBURGH PA 15213 1

DR TAIEB ZNATI
UNIVERSITY OF PITTSBURGH
DEPT OF COMPUTER SCIENCE
PITTSBURGH PA 15260 1

DR MARIE DEJARDINS
SRI INTERNATIONAL
333 RAVENSWOOD AVENUE
MENLO PARK CA 94025 1

MR.ROBERT J. KRUCHTEN 1
HQ AMC/SCA
203 W LOSEY ST, SUITE 1016
SCOTT AFB IL 62225-5223

DR. DAVE GUNNING 1
DARPA/ISO
3701 NORTH FAIRFAX DRIVE
ARLINGTON VA 22203-1714

GINNY ALBERT 1
LOGICON ITG
2100 WASHINGTON BLVD
ARLINGTON VA 22204

ADAM PEASE 1
TECKNOWLEDGE
1810 EMBARCADERO RD
PALO ALTO CA 94303

DR STEPHEN WESTFOLD 1
KESTREL INSTITUTE
3260 HILLVIEW AVE
PALO ALTO CA 94304

DR. STEPHEN E. CROSS, DIRECTOR 1
SOFTWARE ENGINEERING INSTITUTE
CARNEGIE MELLON UNIVERSITY
4500 FIFTH AVE 15213
PITTSBURGH PA 15213

DIRNSA 1
R509
9800 SAVAGE RD
FT MEADE MD 20755-6000

NSA/CSS 1
K1
FT MEADE MD 20755-6000

PHILLIPS LABORATORY 1
PL/TL (LIBRARY)
5 WRIGHT STREET
HANSCOM AFB MA 01731-3004

THE MITRE CORPORATION
D460
202 BURLINGTON ROAD
BEDFORD MA 01732 1

DR. DAVID ETHERINGTON
CIRL, 1269
UNIVERSITY OF OREGON
EUGENE, OR 97403 1

DR. MAREK RUSINKIEWICZ
MICROELECTRONCS & COMPUTER TECH
3500 WEST BALCONES CENTER DRIVE
AUSTIN, TX 78759-6509 1

MAJOR DOUGLAS DYER/ISO
DEFENSE ADVANCED PROJECT AGENCY
3701 NORTH FAIRFAX DRIVE
ARLINGTON, VA 22203-1714 1

DR. STEVE LITTLE
MAYA DESIGN GROUP
2100 WHARTON STREET S&E 702
PITTSBURGH, PA 15203-1944 1

NEAL GLASSMAN
AFOSR
110 DUNCAN AVENUE
BOLLING AFB, WASHINGTON, D.C.
29332 1

AFRL/IFT
525 BROOKS ROAD
ROME, NY 13441-4505 1

AFRL/IFTM
525 BROOKS ROAD
ROME, NY 13441-4505 1

DR. CHARLES L. MOREFIELD
ALPHATECH, INC.
2101 WILSON BLVD, SUITE 402
ARLINGTON VA 22201 1

MR. GARRY W. BARRINGER 1
TECHNICAL DIRECTOR
AEROSPACE CZ ISR CENTER
LANGLEY AFB VA 23665

DR. JAMES HENDLER 1
DEFENSE ADVANCED PROJECT AGENCY
3701 NORTH FAIRFAX DRIVE
ARLINGTON, VA 22203-1714

MISSION OF AFRL/INFORMATION DIRECTORATE (IF)

The advancement and application of information systems science and technology for aerospace command and control and its transition to air, space, and ground systems to meet customer needs in the areas of Global Awareness, Dynamic Planning and Execution, and Global Information Exchange is the focus of this AFRL organization. The directorate's areas of investigation include a broad spectrum of information and fusion, communication, collaborative environment and modeling and simulation, defensive information warfare, and intelligent information systems technologies.